CS 341: ALGORITHMS

Lecture 11: dynamic programming III

Readings: see website

Trevor Brown (co-taught with Anna Lubiw)

https://www.student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca
PROBLEM: LONGEST COMMON SUBSEQUENCE (LCS)

Problem 5.3

Longest Common Subsequence

Instance: Two sequences \( X = (x_1, \ldots, x_m) \) and \( Y = (y_1, \ldots, y_n) \) over some finite alphabet \( \Gamma \).

Find: A maximum length sequence \( Z \) that is a subsequence of both \( X \) and \( Y \).

\[ Z = (z_1, \ldots, z_\ell) \] is a subsequence of \( X \) if there exist indices 
\[ 1 \leq i_1 < \cdots < i_\ell \leq m \] such that \[ z_j = x_{i_j}, \quad 1 \leq j \leq \ell. \]

Similarly, \( Z \) is a subsequence of \( Y \) if there exist (possibly different) indices 
\[ 1 \leq h_1 < \cdots < h_\ell \leq n \] such that \[ z_j = y_{h_j}, \quad 1 \leq j \leq \ell. \]

Let's first solve for the length of the LCS.
EXAMPLES

• $X=\text{aaaaa}$ $Y=\text{bbbb}$ $Z=\text{LCS}(X,Y)=?$
  • $Z=\epsilon$ (empty sequence)
• $X=\text{abcde}$ $Y=\text{bcd}$ $Z=\text{LCS}(X,Y)=?$
  • $Z=\text{bcd}$
• $X=\text{abcde}$ $Y=\text{labeled}$ $Z=\text{LCS}(X,Y)=?$
  • $Z=\text{abe}$
POSSIBLE GREEDY SOLUTIONS?

- Alg: for each $x_i \in X$, try to choose a matching $y_j \in Y$ that is to the right of all previously chosen $y_j$ values

- $X=abcde$  $Y=labef$
- $X=abcde$  $Y=labef$
- $X=abcde$  $Y=labef$ [no suitable $y_j$ found]
- $X=abcde$  $Y=labef$ [no suitable $y_j$ found]
- $X=abcde$  $Y=labef$
- $Z=abe$  Optimal?
POSSIBLE GREEDY SOLUTIONS?

- Alg: for each \( x_i \in X \), try to choose a matching \( y_j \in Y \) that is **to the right** of all previously chosen \( y_j \) values
  - \( X=azbracadabra \) \( Y=abracadabraz \)
  - \( X=azbracadabra \) \( Y=abracadabraz \)
  - \( X=azbracadabra \) \( Y=abracadabraz \) [no \( y_j \) after \( z \)]

Blindly taking \( z \) is bad.

**How to decide** whether to take or leave \( z \)?

**Try both** possibilities!
(Brute force / dynamic programming)

Similar greedy alg that goes right-to-left works for this input, but fails for other inputs.
DEFINING SUBPROBLEMS

• **Full problem**: return $|LCS(X, Y)|$ (i.e., length of LCS)
  
  • Reduce size by taking **prefixes** of $X$ or $Y$
  
  • Let $X_i = (x_1, \ldots, x_i)$ and $Y_i = (y_1, \ldots, y_i)$

<table>
<thead>
<tr>
<th>$X_m$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$\ldots$</th>
<th>$x_{m-1}$</th>
<th>$x_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_4$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Note $X = X_m$ and $Y = Y_n$

• **Subproblem**: return $|LCS(X_i, Y_j)|$

• **Idea for recurrence**: remove the last letter of $X$ or $Y$
Consider optimal solution $Z = \text{LCS}(X, Y)$

Since $x_m, y_n \notin Z$ we know $Z = \text{LCS}(X_{m-1}, Y_{n-1})$

This cannot be the final $a$ in $Z$

Neither of these is part of $Z$

Since $Z$ is a subsequence of $X$, $z_\ell = a$ must appear in $X_{m-1}$

This cannot be the final $a$ in $Z$

$z_\ell = a$ must be in $Y_{n-1}$
**BUILDING SOLUTIONS FROM SUBPROBLEMS**

**EXAMPLE #2**

<table>
<thead>
<tr>
<th>X</th>
<th>a</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>x₂</td>
<td>x₃</td>
<td>x₄</td>
<td>...</td>
<td>xₘ₋₁</td>
<td>xₘ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>a</th>
<th>z</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>d</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>y₁</td>
<td>y₂</td>
<td>y₃</td>
<td>y₄</td>
<td>...</td>
<td>yₙ₋₁</td>
<td>yₙ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Z</th>
<th>a</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>z₁</td>
<td>z₂</td>
<td>z₃</td>
<td>...</td>
<td>zₗ₋₁</td>
<td>zₗ</td>
<td></td>
</tr>
</tbody>
</table>

---

Since $Z$ is a subsequence of $Y$, $z_\ell = a$ must appear in $Y_{n-1}$.

If $x_m \notin Z$, $y_n \in Z$:

$Z = \text{LCS}(X_{m-1}, Y)$

Since $y_n \notin Z$ we know $Z = \text{LCS}(X, Y_{n-1})$.

---

Or maybe this is:

This might be the final $a$ in $Z$.

But this certainly is not! :)

Since $Z$ is a subsequence of $Y$. $z_\ell = a$ must appear in $Y_{n-1}$. 

---
**Example #3**

Then we have $Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell$

**Building Solutions from Subproblems**

Consume $x_m$ and $y_n$ by matching with $z_\ell$
SUMMARIZING CASES

- $z_\ell$ matches **neither** $x_m$ nor $y_n$
  \[ Z = \text{LCS}(X_{m-1}, Y_{n-1}) \]
- $z_\ell$ matches $x_m$ but not $y_n$
  \[ Z = \text{LCS}(X_m, Y_{n-1}) \]
- $z_\ell$ matches $y_n$ but not $x_m$
  \[ Z = \text{LCS}(X_{m-1}, Y_n) \]
- $z_\ell$ matches **both**
  \[ Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell \]
- **... but we don’t know** $z_\ell$
  - Try all cases and maximize
  - **Careful: last case is only valid if** $x_m = y_n$
  - Also note $x_m = y_n$ only holds in the last case
    - Cases 2&3: trivial
    - Case 1: if $x_m = y_n \neq z_\ell$ then we can improve $Z$ (contra)
DERIVING A RECURRENCE

- \( z_\ell \) matches **neither** \( x_m \) nor \( y_n \)  
  \[ (x_m \neq y_n) \quad Z = \text{LCS}(X_{m-1}, Y_{n-1}) \]
- \( z_\ell \) matches \( x_m \) but not \( y_n \)  
  \[ (x_m \neq y_n) \quad Z = \text{LCS}(X_m, Y_{n-1}) \]
- \( z_\ell \) matches \( y_n \) but not \( x_m \)  
  \[ (x_m \neq y_n) \quad Z = \text{LCS}(X_{m-1}, Y_n) \]
- \( z_\ell \) matches **both**  
  \[ (x_m = y_n) \quad Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell \]

- **Let** \( c(i, j) = |\text{LCS}(X_i, Y_j)| \)
- Brainstorming sensible base cases
  - \( i = 0 \) one string is empty, so \( c(0, j) = 0 \) (similarly for \( j = 0 \))
- General cases

\[
\begin{align*}
    c(i, j) &= c(i - 1, j - 1) + 1 & \text{if } x_m = y_n \\
    c(i, j) &= \max\{c(i - 1, j - 1), c(i, j - 1), c(i - 1, j)\} & \text{if } x_m \neq y_n
\end{align*}
\]
RECURRENCE

• Combining expressions

\[ c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i - 1, j - 1) + 1 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i, j - 1), c(i - 1, j), c(i - 1, j - 1)\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j 
\end{cases} \]

• Can simplify!

• Observe \( c(i - 1, j - 1) \leq c(i - 1, j) \)
  (former is a subproblem of the latter)

\[ c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i - 1, j - 1) + 1 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i, j - 1), c(i - 1, j)\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j 
\end{cases} \]
Suppose $X = \text{gdvegta}$ and $Y = \text{gvcekst}$
### Matrix for Comparison

Suppose $X = \text{gdvegta}$ and $Y = \text{gvcekst}$

<table>
<thead>
<tr>
<th></th>
<th>$i = 0$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = 0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>k</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>t</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**Recurrence Relation**

$$c(i,j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
c(i-1,j-1) + 1 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i,j-1), c(i-1,j)\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j 
\end{cases}$$
**Pseudocode**

**Algorithm:** $LCS1(X = (x_1, \ldots, x_m), Y = (y_1, \ldots, y_n))$

for $i \leftarrow 0$ to $m$

$c[i, 0] \leftarrow 0$

for $j \leftarrow 0$ to $n$

$c[0, j] \leftarrow 0$

for $i \leftarrow 1$ to $m$

for $j \leftarrow 1$ to $n$

if $x_i = y_j$

then $c[i, j] \leftarrow c[i - 1, j - 1] + 1$

else $c[i, j] \leftarrow \max\{c[i, j - 1], c[i - 1, j]\}$

return $(c[m, n])$

**Complexity:**

Space? Time? (word RAM model)

$\Theta(nm)$ for both
Computing the LCS
Not Just Its Length

To make it easy to find the actual LCS (not just its length),

Consider which table entry was used to calculate $c[i,j]$

$$c[i,j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i-1,j-1) + 1 & \text{if } i,j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i,j-1), c(i-1,j)\} & \text{if } i,j \geq 1 \text{ and } x_i \neq y_j 
\end{cases}$$

We store the direction to that entry in an array $\pi[i,j]$

Case 1: $c(i,j) = c(i,j-1)$
We store "J" in $\pi[i,j]$ to indicate decrementing $j$ (to get $i,j-1$)

Case 2: $c(i,j) = c(i-1,j)$
We store "I" in $\pi[i,j]$ to indicate decrementing $i$ (to get $i-1,j$)

Case 3: $c(i,j) = c(i-1,j-1) + 1$
We store "IJ" in $\pi[i,j]$ to indicate decrementing both $i$ and $j$

Recall in this case, $x_i = y_j$ so we include $x_i$ in the LCS

In our example table we just draw an arrow to the entry...
SAVING THE DIRECTION TO THE PREDECESSOR SUBPROBLEM $\pi$

```java
LCS2(X[1..m], Y[1..n])
  c = new array[0..m][0..n]
  \pi = new array[0..m][0..n]

  for i = 0..m do c[i][0] = 0
  for j = 0..n do c[0][j] = 0

  for i = 1..m
    for j = 1..n
      if X[i] = Y[j]
        c[i][j] = c[i-1][j-1] + 1
        \pi[i][j] = "IJ"
      else if c[i][j-1] > c[i-1][j]
        c[i][j] = c[i][j-1]
        \pi[i][j] = "J"
      else // c[i][j-1] <= c[i-1][j]
        c[i][j] = c[i-1][j]
        \pi[i][j] = "I"

  return c, \pi
```

**Case:** $c(i, j) = c(i - 1, j - 1) + 1$
We store \texttt{“IJ”} in $\pi[i, j]$ to indicate decrementing both $i$ and $j$

**Case:** $c(i, j) = c(i, j - 1)$
Recall in this case, $x_i = y_j$ so we include $x_i$ in the LCS

**Case:** $c(i, j) = c(i - 1, j)$
We store \texttt{“I”} in $\pi[i, j]$ to indicate decrementing $i$ (to get $i - 1, j$)
Suppose $X = gdvegta$ and $Y = gvcekst$.

How to obtain LCS = gvet from this table?

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>i = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Done: seq = gvet

seq = gvet

seq = et

this is.

seq = t

this “a” is not in

<table>
<thead>
<tr>
<th></th>
<th>seq = et</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
<tr>
<td>k</td>
<td>4</td>
</tr>
<tr>
<td>s</td>
<td>5</td>
</tr>
<tr>
<td>t</td>
<td>6</td>
</tr>
</tbody>
</table>
FindLCS(c[0..m][0..n], π[0..m][0..n], X[0..m])

1. lcs = new string
2. i = m
3. j = n

while i>0 and j>0
  if π[i][j] == "IJ"
    lcs.append(X[i])
    i--
    j--
  else if π[i][j] == "J"
    j--
  else // π[i][j] == "I"
    i--

return reverse(lcs)
PROBLEM: MINIMUM LENGTH TRIANGULATION

- **Input:** \( n \) points \( q_1, \ldots, q_n \) in 2D space that form a **convex** \( n \)-gon \( P \)
  - Assume points are **sorted clockwise** around the center of \( P \)

- **Find:** a triangulation of \( P \) such that the sum of the perimeters of the \( n - 2 \) triangles is minimized

- **Output:** the **sum** of the **perimeters** of the triangles in \( P \)
How many triangulations are there?

Number of triangulations of a convex \( n \)-gon = the \((n - 2)\)nd Catalan number

This is \( C_{n-2} = \frac{1}{n-1} \binom{2n-4}{n-2} \)

It can be shown that \( C_{n-2} \in \Theta(4^n/(n - 2)^{3/2}) \)
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$. 
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:

the triangle $q_1 q_k q_n$, \hspace{1cm} (1)
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:

1. the triangle $q_1 q_k q_n$, \hspace{1cm} (1)
2. the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:

- the triangle $q_1 q_k q_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
- the polygon with vertices $q_k, \ldots, q_n$. \hspace{1cm} (3)
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n-1\}$.

For a given $k$, we have:

- the triangle $q_1 q_k q_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
- the polygon with vertices $q_k, \ldots, q_n$. \hspace{1cm} (3)

The optimal solution will consist of optimal solutions to the two subproblems in (2) and (3), along with the triangle in (1).
PROBLEM DECOMPOSITION

The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:

- the triangle $q_1 q_k q_n$, (1)
- the polygon with vertices $q_1, \ldots, q_k$, (2)
- the polygon with vertices $q_k, \ldots, q_n$. (3)

The optimal solution will consist of optimal solutions to the two subproblems in (2) and (3), along with the triangle in (1).
**RECURSION RELATION**

- Let $S(i, j)$ = optimal solution to the subproblem consisting of the polygon with vertices $q_i \ldots q_j$
- Let $\Delta_{ijk}$ denote $\text{perimeter}(q_i \ldots q_j)$
- If a given point $q_k$ is in the optimal solution, then $S(i, j) = S(i, k) + \Delta_{ijk} + S(k, j)$
RECURRENCE RELATION

• But we don’t know the optimal $k$

• Minimize over all $k$ strictly between $i$ and $j$

$$S(i, k) = 0$$

$$q_i$$  $$S(k, j)$$  $$q_j$$

$$S(i, j) = \begin{cases} 
\min_{i<k<j} \{S(i, k) + \Delta_{i j k} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}$$
FILLING IN THE TABLE

• Table $S[1..n, 1..n]$ of solutions to $S(i, j)$ for all $i, j \in \{1..n\}$

Dependencies:

$S[i, k]$ and $S[k, j]$ 
For $k = (i + 1) \ldots (j - 1)$

$S[i, k]:$ 
$S[i, i + 1] \ldots S[i, j - 1]$

$S[k, j]:$ 
$S[i + 1, j] \ldots S[j - 1, j]$

We depend on larger $i$ 
And same $i$ but smaller $j$

What's a correct fill order? 
for $i = n \ldots 1$, for $j = 1 \ldots n$

\[
S(i, j) = \begin{cases} 
\min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}
\]
$S(i, j) = \begin{cases} 
\min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}$

- Number of subproblems: $n^2$
- Time to solve subproblem $S(i, j)$: $O(j - i) \subseteq O(n)$
- So total runtime is in $O(n^3)$
  - More effort needed to show $\Omega(n^3)$, since so many subproblems are base cases, which take $\Theta(1)$ steps
- **Incidentally**, this is polynomial time (in the input size)
  - But basic runtime analysis does not require such an argument
MEMOIZATION: AN ALTERNATIVE TO DP

Recall that the goal of dynamic programming is to eliminate solving subproblems more than once.

Memoization is another way to accomplish the same goal.

Memoization is a recursive algorithm based on same recurrence relation as would be used by a dynamic programming algorithm.

The idea is to remember which subproblems have been solved; if the same subproblem is encountered more than once during the recursion, the solution will be looked up in a table rather than being re-calculated.

This is easy to do if initialize a table of all possible subproblems having the value undefined in every entry.

Whenever a subproblem is solved, the table entry is updated.
EXAMPLE: USING MEMOIZATION TO COMPUTE FIBONACCI NUMBERS EFFICIENTLY

```plaintext
main
  for i ← 2 to n
do M[i] ← -1
return (RecFib(n))

procedure RecFib(n)
  if n = 0 then f ← 0
  else if n = 1 then f ← 1
  else if M[n] ≠ -1 then f ← M[n]
  else
    \[
    \begin{align*}
    f_1 & \leftarrow \text{RecFib}(n - 1) \\
    f_2 & \leftarrow \text{RecFib}(n - 2)
    \end{align*}
    \]
    \[
    f \leftarrow f_1 + f_2 \\
    M[n] \leftarrow f
    \]
  return (f);
```

If \( M[n] \) is already computed, don’t recurse!
If $M[n]$ is already computed, don’t recurse!

```
procedure RecFib(n)
    if $n = 0$ then $f \leftarrow 0$
    else if $n = 1$ then $f \leftarrow 1$
    else if $M[n] \neq -1$ then $f \leftarrow M[n]$
        $f_1 \leftarrow \text{RecFib}(n - 1)$
        $f_2 \leftarrow \text{RecFib}(n - 2)$
        $f \leftarrow f_1 + f_2$
        $M[n] \leftarrow f$
    else
        $f \leftarrow f_1 + f_2$
    return $(f)$;
```
BONUS CLARIFICATION MATERIAL
MODEL COMPARISON

Word RAM
- Each variable (or array entry, etc.) is a word
- All words have the same size
  - $O(\log W)$ bits where $W = \# \text{ of words in the input (not realistic if variables contain big numbers!)}$
- Runtime is the number of operations on words (where each word operation takes $O(1)$ time)
  - Read/write in $O(1)$
  - Add in $O(1)$
  - Multiply in $O(1)$
- Space complexity is the number of words used (excluding the input)

Bit complexity
- Each variable $x$ is a bit string
- Variables can have different numbers of bits
  - $x$ is encoded in $O(\log x)$ bits
- Runtime is the number of operations on bits (where each bit operation takes $O(1)$ time)
  - Read/write in $O(\log x)$ ($\log x = \# \text{ bits in } x$)
  - Add $x + y$ in $O(\log x + \log y)$
  - Multiply in $O(\log x \times \log y)$
- Space complexity is the number of bits used (excluding the input)

Unit cost model (not used anymore)
- Words have unlimited size
- But still $O(1)$ access time
CALCULATIONS USING INPUT SIZE

• Clarification: you can compute space/time complexity without calculating the input size

• Input size calculations are typically only needed if we ask you to show an algorithm runs in polytime
  • “runs in polytime” means the runtime is at most a polynomial in the number of bits in the input
  • Lots of this in tractability / NP completeness

• Just trying to expose you to these ideas ahead of time…
SO... IS DP LCS A POLYTIME ALGORITHM?

• Is $nm$ polynomial in the input size (# of bits in the input)?
• Word RAM model
  • Input contains $\Theta(n + m)$ words
  • Word RAM model says each word stores $\Theta(\log w)$ bits
    where $w = \#$ words in the input
      • So in this case, $\Theta(\log(n + m))$ bits per word
  • So $S \in \Theta((n + m) \log(n + m))$
    • Want a term that looks like $nm$
SO... IS DP LCS A POLYTIME ALGORITHM?

• Try squaring: $S^2 \in \Theta((n + m) \log(n + m)^2)$
• $\Theta(n^2 \log^2(n + m) + nm \log^2(n + m) + m^2 \log^2(n + m))$
• Of course, $nm \in O(nm \log^2(n + m))$
  • ... which is just one of the terms of $S^2$
  • So $nm \in O(S^2)$
• So the runtime is polynomial in the input size (in bits)
  • But this was ugly. Is there a simpler approach?
SO... IS DP LCS A POLYTIME ALGORITHM?

- Calculation using words would be simpler...
- **Words** in the input $W \in O(n + m)$
- $W^2 \in O((n + m)^2) = O(n^2 + nm + m^2)$
  - So $O(nm) \subseteq O(W^2)$
  - I.e., polynomial in the # of **words** in the input

- How does this help us?
  - # **words** $W$ in the input $\leq$ # **bits** $S$ in the input
  - So $O(nm) \subseteq O(W^2)$ **implies** $O(nm) \subseteq O(S^2)$

So, for showing **polytime**, you can calculate the input size using **words** (and justify like this)