CS 341: ALGORITHMS

Lecture 11: dynamic programming III

Readings: see website

Trevor Brown

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

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

Let's first solve for the length of the 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 $z_j = x_{i_j}$, $1 \leq j \leq \ell$.

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

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

- $X=\text{aaaaa} \quad Y=\text{bbbb} \quad Z=LCS(X,Y)=? \quad Z=\varepsilon$ (empty sequence)
- $X=\text{abcde} \quad Y=\text{bcd} \quad Z=LCS(X,Y)=? \quad Z=\text{bcd}$
- $X=\text{abcde} \quad Y=\text{labeled} \quad Z=LCS(X,Y)=? \quad Z=\text{abe}$
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

  - $X=\text{azbracadabra}$, $Y=\text{a} \text{bracadabraz}$
  - $X=\text{azbracadabra}$, $Y=\text{a} \text{bracadabra} \text{z}$
  - $X=\text{azbracadabra}$, $Y=\text{a} \text{bracadabraz}$ [no $y_j$ after $z$
  - $X=\text{azbracadabra}$, $Y=\text{a} \text{bracadabraz}$ [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 $|\text{LCS}(X, Y)|$ (i.e., length of LCS)
- Reduce size by taking *prefixes* of $X$ or $Y$
- Let $X_i = (x_1, ..., x_i)$ and $Y_i = (y_1, ..., y_i)$

<table>
<thead>
<tr>
<th>$X_m$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>...</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 $|\text{LCS}(X_i, Y_j)|$
- **Idea for recurrence:** remove the last letter of $X$ or $Y$
Neither of these is part of $Z$

Consider optimal solution $Z = LCS(X, Y)$

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

EXAMPLE #2

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

Or maybe this is

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

But this certainly is not :)

Case $x_m \notin Z, y_n \in Z$ is symmetric

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

This might be the final \( a \) in \( Z \)

Or maybe this is...

This might be the final \( a \) in \( Z \)

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 $y_n$ \( (x_m \neq y_n) \) $Z = \text{LCS}(X_{m-1}, Y_{n-1})$
- $z_\ell$ matches $x_m$ but not $y_n$ \( (x_m \neq y_n) \) $Z = \text{LCS}(X_m, Y_{n-1})$
- $z_\ell$ matches $y_n$ but not $x_m$ \( (x_m \neq y_n) \) $Z = \text{LCS}(X_{m-1}, Y_n)$
- $z_\ell$ matches **both** \( (x_m = y_n) \) $Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell$

- 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

<table>
<thead>
<tr>
<th>$c(i, j)$</th>
<th>[c(i - 1, j - 1) + 1]</th>
<th>$c(i, j) = \max{c(i - 1, j - 1), c(i, j - 1), c(i - 1, j)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if $x_m = y_n$</td>
<td>if $x_m \neq y_n$</td>
</tr>
</tbody>
</table>

Recall $Z = \text{LCS}(X_m, Y_n)$
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 $Y = \text{gvcekst}$

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

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

<table>
<thead>
<tr>
<th></th>
<th>$X$</th>
<th>$g$</th>
<th>$d$</th>
<th>$v$</th>
<th>$e$</th>
<th>$g$</th>
<th>$t$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
PSEUDOCODE

Algorithm: LCS1 (X = (x₁, ..., xₘ), Y = (y₁, ..., yₙ))

for i ← 0 to m
    c[i, 0] ← 0
for j ← 0 to n
    c[0, j] ← 0
for i ← 1 to m
    for j ← 1 to n
        if xᵢ = yⱼ
            then c[i, j] ← c[i - 1, j - 1] + 1
        else c[i, j] ← max{c[i, j - 1], c[i - 1, j]}
return (c[m, n]);

Complexity:
Space? Time?
(word RAM model)

\[ \Theta(nm) \text{ for both} \]
In our example table we just draw an arrow to the entry...

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
1  LCS2(X[1..m], Y[1..n])
2  c = new array[0..m][0..n]
3  $\pi$ = new array[0..m][0..n]
4
5  for i = 0..m do c[i][0] = 0
6  for j = 0..n do c[0][j] = 0
7
8  for i = 1..m
9    for j = 1..n
10      if X[i] = Y[j]
11        c[i][j] = c[i-1][j-1] + 1
12        $\pi[i][j] = "IJ"
13      else if c[i][j-1] > c[i-1][j]
14        c[i][j] = c[i][j-1]
15        $\pi[i][j] = "J"
16      else // c[i][j-1] <= c[i-1][j]
17        c[i][j] = c[i-1][j]
18        $\pi[i][j] = "I"
19
20  return c, $\pi$
```

Case: $c(i, j) = c(i - 1, j - 1) + 1$

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

Case: $c(i, j) = c(i, j - 1)$

We store “J” in $\pi[i, j]$ to indicate decrementing $j$ (to get $i, j - 1$)

Case: $c(i, j) = c(i - 1, j)$

We store “I” in $\pi[i, j]$ to indicate decrementing $i$ (to get $i - 1, j$)
Suppose $X = gdvegta$ and $Y = gveckst$.

**Example**

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

Done: seq=gvet

seq=gvet

seq=gvet

seq=et

this is.

seq=t

this “a” is not in

How to obtain LCS=gvet from this table?
FOLLOWING PREDECESSORS TO COMPUTE THE LCS

Complexities of this trace-back algo:
Space? Time?
(word RAM model)

space: \(O(n+m)\) words

time: \(O(n+m)\)

```python
FindLCS(c[0..m][0..n], \pi[0..m][0..n], X[0..m])
    lcs = new string
    i = m
    j = n
    while i>0 and j>0
        if \pi[i][j] == "IJ"
            lcs.append(X[i])
            i--
            j--
        else if \pi[i][j] == "J"
            j--
        else // \pi[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 $C_{n-2} \in \Theta(4^n/(n-2)^{3/2})$
PROBLEM DECOMPOSITION

The edge $q_nq_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$. 
PROBLEM DECOMPOSITION

The edge $q_nq_1$ 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 $q_1q_kq_n$, \hspace{1cm} (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 $q_1 q_k q_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
PROBLEM DECOMPOSITION

The edge $q_nq_1$ 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 $q_1q_kq_n$, \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)
PROBLEM DECOMPOSITION

The edge $q_nq_1$ 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 $q_1q_kq_n$, \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 $q_1 q_k q_n$, \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).
RECURRENCE RELATION

- Let $S(i, j) = \text{optimal solution to the subproblem consisting of the polygon with vertices } q_i \ldots q_j$

- Let $\Delta_{ijk}$ denote $\text{perimeter}(q_i \triangle q_j q_k)$

- If a given triangle $q_i, q_j, q_k$ is in the optimal solution, then $S(i, j) = S(i, k) + \Delta_{ijk} + S(k, j)$
But we don’t know the optimal $k$

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

$$S(i, j) = \begin{cases} \min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\ 0 & \text{otherwise} \end{cases}$$
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 $k = (i + 1) \ldots (j - 1)$

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

- What’s a correct fill order?
  - for $i = n..1$, for $j = 1..n$

- $S(i, j) = \begin{cases} 
\min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}$
RUNTIME
WORD RAM MODEL

- 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

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

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 $f ← M[n]$
    do $f_1 ← RecFib(n - 1)$
        $f_2 ← RecFib(n - 2)$
    else
      do $f ← f_1 + f_2$
        $M[n] ← f$
  return $(f)$;

If $M[n]$ is already computed, don't recurse!
VISUALIZING MEMOIZATION

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 $f \leftarrow M[n]$
    \begin{align*}
    f_1 & \leftarrow \text{RecFib}(n - 1) \\
    f_2 & \leftarrow \text{RecFib}(n - 2)
    \end{align*}
else
    \begin{align*}
    f & \leftarrow f_1 + f_2 \\
    M[n] & \leftarrow f
    \end{align*}
return ($f$);
BONUS CLARIFICATION MATERIAL
MODEL COMPARISON

Word RAM
- Each variable (or array entry, etc.) \( x \) is a word
- All words have the same size
  - \( O(\log W) \) bits where \( W \) = # 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 $w = \# \text{ words in the input}$
    - 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).