THIS TIME

• DP: longest common subsequence (partially covered)
• Memoization VS dynamic programming
• DP: minimum length triangulation
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 $z_j = x_{i_j}, 1 \leq j \leq \ell$.

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

- $X=\text{aaaaa}$       $Y=\text{bbbb}$       $Z=\text{LCS}(X,Y)=?$
  - $Z=\epsilon$ (empty sequence)
- $X=\text{abcde}$       $Y=\text{bcd}$       $Z=\text{LCS}(X,Y)=?$
  - $Z=\text{bcd}$
- $X=\text{abcde}$       $Y=\text{labef}$       $Z=\text{LCS}(X,Y)=?$
  - $Z=\text{abe}$
### THINKING ABOUT SUBPROBLEMS

- Entire problem: # characters in LCS(X,Y)
- How to reduce problem size? Reduce size of X or Y.
- Define X’ and Y’ as follows

<table>
<thead>
<tr>
<th>X =</th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
<th>...</th>
<th>xₓ</th>
<th>xₓ₋₁</th>
<th>xₓ⁺₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’ =</td>
<td>x₁</td>
<td>x₂</td>
<td>x₃</td>
<td>x₄</td>
<td>...</td>
<td>xₓ₋₁</td>
<td>xₓ⁺₁</td>
<td></td>
</tr>
<tr>
<td>Y =</td>
<td>y₁</td>
<td>y₂</td>
<td>y₃</td>
<td>y₄</td>
<td>...</td>
<td>yₓ₋₁</td>
<td>yₓ</td>
<td></td>
</tr>
<tr>
<td>Y’ =</td>
<td>y₁</td>
<td>y₂</td>
<td>y₃</td>
<td>y₄</td>
<td>...</td>
<td>yₓ₋₁</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CONSIDER AN OPTIMAL SOLUTION Z

Can we express Z in terms of X' and Y' instead of X and Y?

- By definition, $Z = \text{LCS}(X, Y)$

<table>
<thead>
<tr>
<th>$Z =$</th>
<th>$z_1$</th>
<th>$z_2$</th>
<th>...</th>
<th>$z_{\ell-1}$</th>
<th>$z_{\ell}$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$X =$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>...</th>
<th>...</th>
<th>...</th>
<th>$x_{m-1}$</th>
<th>$x_m$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$X' =$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>...</th>
<th>...</th>
<th>...</th>
<th>$x_{m-1}$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$Y =$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>...</th>
<th>$y_{n-1}$</th>
<th>$y_n$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$Y' =$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>...</th>
<th>$y_{n-1}$</th>
</tr>
</thead>
</table>
CONSIDER AN OPTIMAL SOLUTION $Z$

- By definition, $Z = \text{LCS}(X,Y)$
- Suppose $z_\ell$ matches both $x_m$ and $y_n$

$Z = \begin{bmatrix} z_1 & z_2 & \ldots & z_{\ell-1} & z_\ell \end{bmatrix}$

$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & \ldots & x_{m-1} & x_m \end{bmatrix}$

$X' = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & \ldots & x_{m-1} \end{bmatrix}$

$Y = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & \ldots & y_{n-1} & y_n \end{bmatrix}$

$Y' = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & \ldots & y_{n-1} \end{bmatrix}$

Can we express $Z$ in terms of $X'$ and $Y'$ instead of $X$ and $Y$?

Then $Z = \text{LCS}(X',Y') + z_\ell$

Consumed by being matched with $y_n$

Consumed by being matched with $x_m$
**CONSIDER AN OPTIMAL SOLUTION Z**

<table>
<thead>
<tr>
<th>Z</th>
<th>z₁</th>
<th>z₂</th>
<th>...</th>
<th>z_{ℓ−1}</th>
<th>z_{ℓ}</th>
</tr>
</thead>
</table>

- By definition, $Z = \text{LCS}(X, Y)$
- Suppose $z_{ℓ}$ matches only $x_{m}$ (so $x_{m} \neq y_{n}$)

<table>
<thead>
<tr>
<th>X</th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
<th>...</th>
<th>x_{m−1}</th>
<th>x_{m}</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'</td>
<td>x₁</td>
<td>x₂</td>
<td>x₃</td>
<td>x₄</td>
<td>...</td>
<td>x_{m−1}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>y₁</th>
<th>y₂</th>
<th>y₃</th>
<th>y₄</th>
<th>...</th>
<th>y_{n−1}</th>
<th>y_{n}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y'</td>
<td>y₁</td>
<td>y₂</td>
<td>y₃</td>
<td>y₄</td>
<td>...</td>
<td>y_{n−1}</td>
<td></td>
</tr>
</tbody>
</table>

Can we express $Z$ in terms of $X'$ and $Y'$ instead of $X$ and $Y$?

Then $Z = \text{LCS}(X, Y')$

(Might be matched with something in $Y'$)

**Maybe** still needed by $Z$

Not needed by $Z$

Remove to shrink problem size!
CONSIDER AN OPTIMAL SOLUTION Z

• By definition, $Z = LCS(X,Y)$
• Suppose $z_\ell$ matches only $y_n$ (so $x_m \neq y_n$)

Can we express $Z$ in terms of $X'$ and $Y'$ instead of $X$ and $Y$?

Then $Z = LCS(X', Y)$

Not needed by $Z$

Maybe still needed by $Z$
CONSIDER AN **OPTIMAL SOLUTION** Z

- By definition, $Z = LCS(X, Y)$
- Suppose $z_\ell$ matches neither.

**Note that** $x_m \neq y_n$, or else we could improve $Z$ by adding them!

Can we express $Z$ in terms of $X'$ and $Y'$ instead of $X$ and $Y$?

Take $Z = LCS(X', Y')$

Not needed by Z

Not needed by Z
FOUR CASES

- Case $z_\ell$ matches both (so $x_m = y_n$): $Z = LCS(X', Y') + z_\ell$
- Case $z_\ell$ matches only $x_m$ (so $x_m \neq y_n$): $Z = LCS(X, Y')$
- Case $z_\ell$ matches only $y_n$ (so $x_m \neq y_n$): $Z = LCS(X', Y)$
- Case $z_\ell$ matches neither (recall $x_m \neq y_n$): $Z = LCS(X', Y')$

- We don’t know $z_\ell$! How to identify case 1 vs 2-4? (If $x_m = y_n$)
- How to differentiate between cases 2-4 without knowing $z_\ell$?

- Try all 3 possibilities in the recurrence and maximize length!

Let $X_i = (x_1, ..., x_i), Y_j = (y_1, ..., y_j)$ and $c[i, j] = |LCS(X_i, Y_j)|$

In-class exercise: derive the recurrence for $c[i, j]$ (part 1) and give pseudocode to solve the problem (part 2)
IN-CLASS EXERCISE PART 1: DERIVE \( c[i, j] \)

- Case \( z_{\ell} \) matches both (so \( x_m = y_n \)): \( Z = LCS(X', Y') + z_{\ell} \)
- Case \( z_{\ell} \) matches only \( x_m \) (so \( x_m \neq y_n \)): \( Z = LCS(X, Y') \)
- Case \( z_{\ell} \) matches only \( y_n \) (so \( x_m \neq y_n \)): \( Z = LCS(X', Y) \)
- Case \( z_{\ell} \) matches neither (recall \( x_m \neq y_n \)): \( Z = LCS(X', Y') \)

Let \( X_i = (x_1, ... , x_i), Y_j = (y_1, ... , y_j) \) and \( c[i, j] = |LCS(X_i, Y_j)| \)

\( c[i, j] = \begin{cases} ? \quad \text{if } i = 0 \text{ or } j = 0 \\ ? \quad \text{if } i, j \geq 1 \text{ and } x_i = y_j \\ ? \quad \text{if } i, j \geq 1 \text{ and } x_i \neq y_j \end{cases} \)
IN-CLASS EXERCISE \textbf{PART 1: DERIVE } c[i, j]

- Case \( z_{\ell} \) matches both (so \( x_m = y_n \)): \( Z = LCS(X', Y') + z_{\ell} \)
- Case \( z_{\ell} \) matches only \( x_m \) (so \( x_m \neq y_n \)): \( Z = LCS(X, Y') \)
- Case \( z_{\ell} \) matches only \( y_n \) (so \( x_m \neq y_n \)): \( Z = LCS(X', Y) \)
- Case \( z_{\ell} \) matches neither (recall \( x_m \neq y_n \)): \( Z = LCS(X', Y') \)

Let \( X_i = (x_1, ..., x_i), Y_j = (y_1, ..., y_j) \) and \( c[i, j] = |LCS(X_i, Y_j)| \)

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

\textbf{Can simplify!} Observe that \( c[i - 1, j - 1] \leq c[i, j - 1] \), because the former only has a \textbf{subset} of the input to the latter!

\textbf{Therefore, it \textbf{can't} be the max}
IN-CLASS EXERCISE PART 1: DERIVE $c[i, j]$

- Case $z_\ell$ matches both (so $x_m = y_n$): $Z = LCS(X', Y') + z_\ell$
- Case $z_\ell$ matches only $x_m$ (so $x_m \neq y_n$): $Z = LCS(X, Y')$
- Case $z_\ell$ matches only $y_n$ (so $x_m \neq y_n$): $Z = LCS(X', Y)$
- Case $z_\ell$ matches neither (recall $x_m \neq y_n$): $Z = LCS(X', Y')$

Let $X_i = (x_1, \ldots, x_i)$, $Y_j = (y_1, \ldots, y_j)$ and $c[i, j] = |LCS(X_i, Y_j)|$

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

$$c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max\{c[i, j - 1], c[i - 1, j]\} + 1 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c[i, j - 1], c[i - 1, j]\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j 
\end{cases}$$
EXERCISE PART 2: PSEUDOCODE

Give pseudocode to compute $c[i,j]$ for all $i,j$ and return the length of LCS($X,Y$).

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

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

Complexity? Space? Time?

$\Theta(nm)$ for both

Start here: ???

Return: ???
COMPUTING THE LCS (NOT ITS LENGTH)

To make it easy to find the actual LCS (not just its length), we keep track of three possible cases that can arise in the recurrence relation:

**UL** $x_i = y_j$, include this symbol in the LCS, denote by $\backslash$

**L** $x_i \neq y_j$, $c[i, j - 1] > c[i - 1, j]$, denote by $\leftarrow$

**U** $x_i \neq y_j$, $c[i, j - 1] \leq c[i - 1, j]$, denote by $\uparrow$

As a mnemonic aid, **U** denotes “up” and **L** denotes “left”.

$$c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
2 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j
\end{cases}$$
SAVING THE DIRECTION TO THE **PREDECESSOR** SUBPROBLEM $\pi$

**Algorithm:** $LCS2(X = (x_1, \ldots, x_m), Y = (y_1, \ldots, y_n))$

1. for $i \leftarrow 0$ to $m$ do $c[i, 0] \leftarrow 0$
2. for $j \leftarrow 0$ to $n$ do $c[0, j] \leftarrow 0$
3. 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 if $c[i, j-1] > c[i-1, j]$
       - then $c[i, j] \leftarrow c[i, j-1]$
     - else $c[i, j] \leftarrow c[i-1, j]$

4. return $(c, \pi)$;

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

+1 means $x_i$ is in the LCS!

If there are **multiple** possible sequences with the **same length** $|LCS(X,Y)|$ then $x_i$ is in **some** such sequence.
Suppose $X = gdvegta$ and $Y = gvcekst$.

**Example**

<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>$1$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>$\uparrow$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
<td>$\leftarrow$</td>
<td>$\uparrow$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**How to obtain LCS=gvet from this table?**

- seq=gvet
- seq=gvet
- seq=gvet
- seq=vet
- seq=et
- seq=t
- this "a" is not in
- this is.
- Done: seq=gvet
Algorithm: \( \text{FindLCS}(c, \pi, v) \)
\[
\begin{align*}
\text{seq} & \leftarrow () \\
i & \leftarrow m \\
j & \leftarrow n \\
\text{while} & \min\{i, j\} \geq 0 \\
& \begin{cases}
\text{if} \ \pi[i, j] = \text{UL} \\
\text{then} & \begin{cases}
\text{seq} & \leftarrow x_i || \text{seq} \\
i & \leftarrow i - 1 \\
j & \leftarrow j - 1
\end{cases}
\end{cases}
\end{align*}
\]
\[
\text{else if} \ \pi[i, j] = \text{L} & \begin{cases}
j & \leftarrow j - 1
\end{cases}
\]
\[
\text{else} & \begin{cases}
i & \leftarrow i - 1
\end{cases}
\]
\text{return} \ (\text{seq})

Complexity of this trace-back:
Space? Time?
Recall: \(|X| = m, |Y| = n\)

space: \(O(nm)\)
time: \(O(n+m)\)
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 MEMORIZATION 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]
        \{ f1 ← RecFib(n − 1) \\
        f2 ← RecFib(n − 2) \}
    else \{ f ← f1 + f2 \\
               M[n] ← f \}
    return (f);
```
If $M[n]$ is already computed, don't recurse!

Algorithm: $BadFib(n)$
if $n = 0$ then $f \leftarrow 0$
else if $n = 1$ then $f \leftarrow 1$
else \{ $f_1 \leftarrow BadFib(n - 1)$ $f_2 \leftarrow BadFib(n - 2)$ $f \leftarrow f_1 + f_2$ \}
return $(f)$;

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 $f \leftarrow M[n]$ \{ $f_1 \leftarrow RecFib(n - 1)$ $f_2 \leftarrow RecFib(n - 2)$ $f \leftarrow f_1 + f_2$ \}
else \{ $M[n] \leftarrow f$ \}
return $(f)$;

Memoization reduces this tree to a line with right-hanging leaves.
# recursive calls = $O(n)$ instead of $\sim 2^n$

Calls not needed because of memoization
PROBLEM: MINIMUM LENGTH TRIANGULATION

• Input: \( n \) points \( q_1, \ldots, q_n \) in 2D space that form a convex \( n \)-gon \( P \)

• 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 \( P \)

Input points are sorted in clockwise order around the center of \( P \)

[Example input on blackboard]
### How many triangulations are there?

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

### Number of triangulations of a convex $n$-gon

$n$-gon = the $(n - 2)$th Catalan number

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 $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:
- the triangle $q_1q_kq_n$,
- the polygon with vertices $q_1, \ldots, q_k$,
- the polygon with vertices $q_k, \ldots, q_n$.

The optimal solution will consist of optimal solutions to the two subproblems in (2) and (3), along with the triangle in (1).
For $1 \leq i < j \leq n$, let $S[i, j]$ denote the optimal solution to the subproblem consisting of the polygon having vertices $q_i, \ldots, q_j$.

Let $\Delta(q_i, q_k, q_j)$ denote the perimeter of the triangle having vertices $q_i, q_k, q_j$.

The we have the recurrence relation

$$S[i, j] = \min \{ \Delta(q_i, q_k, q_j) + S[i, k] + S[k, j] : i < k < j \}.$$ 

The base cases are given by

$$S[i, i + 1] = 0$$

for all $i$.

How to fill in the table? [blackboard]
NEXT TIME

• **Graph** algorithms

• *Maybe:* **big-picture** overview of the algorithmic design paradigms we’ve seen so far
  • Brute force, divide and conquer, dynamic programming, greedy
  • Pros/cons of each? When to use each?