But we will discover this in a future computation.

ex: For \( \overline{X}_6 = B A C D N E \)

\[ \begin{array}{c}
\overline{X}_5 \\
\overline{Y}_3 = B N E N \\
\end{array} \]

| LCS1 = 1 | new LCS = 2 |

LCS \((\overline{X}_6, \overline{Y}_3)\) computed after LCS \((\overline{X}_5, \overline{Y}_4)\).

Would match current \(N(x_5)\) with a previous \(N(y_2)\)

only if there is some future char in \(X\) that can be matched with a char in \(Y\)
(e.g. \(E\) in the example above).

But we would discover this in a future computation - e.g. when computing LCS \((\overline{X}_6, \overline{Y}_3)\)
in the example above (the matching \(E\)'s).

Case 2: \(x_i \neq y_j\)

\[ \begin{array}{c}
X = B A C D E \\
Y = B D E B \\
\end{array} \]

\(LCS\) won't improve (from a previous computed LCS).

3 choices we can make for LCS \((\overline{X}_i, \overline{Y}_j)\)
3 choices we can make:

- **A**
  \[ \text{LCS}(\overline{x_i}, \overline{y_j}) \]
  ends in \( x_i \)
  
  \[ \begin{array}{c}
  x = \boxed{E} \\
  y = \boxed{E} \end{array} \]

- **B**
  \[ \text{LCS}(\overline{x_i}, \overline{y_j}) \]
  ends in \( y_j \)
  
  \[ \begin{array}{c}
  x = \boxed{} \\
  y = \boxed{} \end{array} \]

- **C**
  \[ \text{LCS} \text{ ends in neither } x_i \text{ nor } y_j \]
  
  \[ \begin{array}{c}
  x = \boxed{K} \\
  y = \boxed{J} \end{array} \]

\[ \text{LCS}(\overline{x_i}, \overline{y_j}) = \text{LCS}(\overline{x_i}, \overline{y_{j-1}}) \]
\[ \text{LCS}(\overline{x_{i-1}}, \overline{y_j}) \]
\[ \text{LCS}(\overline{x_{i-1}}, \overline{y_{j-1}}) \]

These LCS have already been computed!

But LCS can't end in both \( x_i, y_j \):

\[ \begin{array}{c}
  x = \boxed{BACDE} \\
  y = \boxed{BDE} \end{array} \]

\[ \text{LCS}(\overline{x_i}, \overline{y_j}) = \max \left( \begin{array}{c}
  \text{LCS}(\overline{x_i}, \overline{y_{j-1}}), \text{LCS}(\overline{x_{i-1}}, \overline{y_j}), \text{LCS}(\overline{x_{i-1}}, \overline{y_{j-1}}) \end{array} \right) \]

- **A**
- **B**
- **C**

will always be \( \leq \text{(computed before A, B)} \)

(ex if \( x = \boxed{BACDE} \) Better to match \( y = \boxed{BDE} \) than \( B \)

matching:

- **E**:
  \[ \text{LCS}(X, Y) = \text{DE} \]
- **B**:
  \[ \text{LCS}(X, Y) = B \]
Dynamic Programming Solution

Recall: \( X \) is length \( m \), \( Y \) is length \( n \)

(1) Size of array to store values? \( \max((m+1) \times (n+1)) \)
   Compare every prefix of \( X \) to every prefix of \( Y \).

(2) What each entry holds?
   \( c[i][j] = \text{LCS}(X_i, Y_j) \) (length).

(3) Dynamic Programming Formulation:
   \[
   c[i][j] = \begin{cases} 
   0 & \text{if } i = 0 \text{ or } j = 0 \\
   c[i-1][j-1] + 1 & \text{if } X_i = Y_j \\
   \max(c[i-1][j], c[i][j-1]) & \text{if } X_i \neq Y_j
   \end{cases}
   \]

(4) Base Case? \( i = 0 \) or \( j = 0 \).

(5) To fill matrix:
   \( c[0][0..n] = 0 \), \( c[0..m][0] = 0 \)
   
   For (\( i = 1 \) to \( m \)) // every prefix of \( X \)
   For (\( j = 1 \) to \( n \)) // every prefix of \( Y \)
     \( c[i][j] = \text{<based on D.P. formulation>} \)

(6) Optimal value (length)? In \( c[m][n] \).

(7) Run Time? \( O(mn) \).
How to find actual LCS?

Quick example: ...
Actual solution (LCS)?

Quick example will show how to find subsequence.

ex 1: \( X = B A C D B \) \( m = 5 \) \( n = 4 \)
\( Y = B D E B \)

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
& 0 & B & D & E & B \\
\hline
0 & 0 & * & 1 & 1 & 1 & * \\
\hline
B & 0 & 1 & 1 & 1 & 1 & 1 \\
\hline
A & 0 & 1 & 1 & 1 & 1 & 1 \\
\hline
C & 0 & 1 & 1 & 1 & 1 & 1 \\
\hline
D & 0 & 1 & 2 & 2 & 2 & 2 \\
\hline
B & 0 & 1 & 2 & 2 & 2 & 3 \\
\hline
\end{array}
\]

Fill in row-order

arbitrarily break ties
between \( \leftarrow \) and \( \uparrow \) by
choosing \( \leftarrow \)

Find subsequence?

Keep 2D array of arrows that indicate where
best LCS was from.

Notice: Only indices with \( \leftarrow \) indicate that corresponding
char is in LCS. So store these chars.

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
\hline
0 & \phi & - & - & - & - \\
\hline
1 & B & \leftarrow & \leftarrow & \leftarrow & \leftarrow \\
\hline
2 & A & \leftarrow & \leftarrow & \leftarrow & \leftarrow \\
\hline
3 & C & \uparrow & \leftarrow & \leftarrow & \leftarrow \\
\hline
4 & D & \uparrow & \leftarrow & \leftarrow & \leftarrow \\
\hline
5 & B & \uparrow & \leftarrow & \leftarrow & \leftarrow \\
\hline
\end{array}
\]

LCS: \( B D B \)
GetSubsequence()
1. Start at $C[i][n]$
2. Follow arrows until $-$ (null char)
3. If ever find $\uparrow$ arrow, output corresponding character (if $C[i][j] = \uparrow$, output $x_i$ (or $y_j$)
4. Return reverse of outputted string
LCS Problem

another example...

\[ X = B C A A D \quad m = 5 \quad n = 4 \]
\[ Y = A A C D \]

```
   D A A C D
  ------------------
  D         0     0     0     0
  B         0     0     0     0
  C         0     0     0     *1 C ← 1
  A         0     1     1     1
  A         1     1     1     1
  D         1     2     2     2     3 D
```

"incorrect" match

Find sequence:
Start at \([m][n]\)
For every \(i\) output character (in reverse)

\[ \text{LCS}(X_5, Y_4) = - A A D \]
Find actual subsequences? Quick ex.

ex: \(X = BCEAD\), \(m = 5\), \(n = 3\)

\(Y = ADA\)

\[\begin{array}{ccccc}
O & A & D & A & \\
0 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 0 \\
A & 0 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0 \\
\end{array}\]

1. Note: First A's in \(X, Y\) matched
2. although "wrong" A's matched later
3. where we correctly decide that matching the first A in \(X\) yields longer LCS

How to find actual LCS?

- Keep arrows that indicate where best LCS was from.
- Notice: only indices with ^ indicate that corresponding char is in LCS. So store these chars.

\[\begin{array}{ccccc}
O & A & D & A & \\
D & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 0 \\
A & 0 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0 \\
\end{array}\]

Break ties between \(\leftarrow, \uparrow\) by arbitrariness choosing \(\leftarrow\)

1. Start at \(C[m][n]\), \(\text{ex: } \leftarrow \uparrow \uparrow\)
2. Follow arrows
3. If see '^' output character
4. LCS is reverse of outputted string

LCS = AD
Quick example will show how to find subsequence.

ex(1) \( X = BCEAD \) \( m = 5 \) \( n = 4 \)

\( Y = BDEB \)

<table>
<thead>
<tr>
<th></th>
<th>( B )</th>
<th>( D )</th>
<th>( E )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( B )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( C )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( E )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( A )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( D )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>