Given two sequences $X = (x_1, x_2, ..., x_m)$ and $Y = (y_1, y_2, ..., y_n)$, $Z$ is a common subsequence of $X$ and $Y$ of length $k$ if there are two strictly increasing sequence of indices $i$ and $j$ such that for $p = 1, 2, \ldots, k$, $x_{i_p} = y_{j_p} = z_p$.

Example:
$X$: A B C B D A B
$Y$: B D C A B A
$Z$: B C B A

Problem: Find a longest common subsequence (lcs) of $X$ and $Y$ in $O(mn)$ time
Solution: Use Dynamic Programming
Step 1: Space of Subproblems

For $1 \leq i \leq m$, and $1 \leq j \leq n$,

- Define $d_{i,j}$ to be the length of the longest common subsequence of $X[1..i]$ and $Y[1..j]$.
- Let $D$ be the $m \times n$ matrix $[d_{i,j}]$. 
Step 2: Recursive Formulation

Let $Z_k = (z_1, \ldots, z_k)$ be a LCS of $X[1..i]$ and $Y[1..j]$.

Case 1: If $x_i = y_j$, then $z_k = x_i = y_j$ and $Z_k$ is $\text{LCS}(X[1..i-1], Y[1..j-1])$ followed by $z_k$.

Case 2: If $x_i \neq y_j$, then $Z_k$ is either $\text{LCS}(X[1..i-1], Y[1..j])$ or $\text{LCS}(X[1..i], Y[1..j-1])$.

If $x_i \neq y_j$, the answer is the larger of the LCS’s of those two cases.

$$d_{i,j} = \begin{cases} 
  d_{i-1,j-1} + 1 & \text{if } x_i = y_j \\
  \max\{d_{i-1,j}, d_{i,j-1}\} & \text{if } x_i \neq y_j 
\end{cases}$$
Step 3: Bottom-up Computation by Rows

Initialize first row and column of the matrix \( (d[0, j] \text{ and } d[i, 0]) \) to 0

Calculate \( d[1, j] \) for \( j = 1, 2, \ldots, n \)

Then, \( d[2, j] \) for \( j = 1, 2, \ldots, n \)

Then, \( d[3, j] \) for \( j = 1, 2, \ldots, n \)

....

We fill the table row by row, filling in each row, left to right.

<table>
<thead>
<tr>
<th>( D[i, j] )</th>
<th>( j = 0 )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>\cdots</th>
<th>\cdots</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\cdots</td>
<td>\cdots</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( m )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution

We also create another $m \times n$ matrix $p[i,j]$ for $1 \leq i \leq m$, and $1 \leq j \leq n$.
This stores which of the three choices led to the maximum value creating $d[i,j]$.
This is done by pointing an arrow towards the entry that led to that choice. These arrows will permit reconstructing the elements of the LCS.
LONGEST-COMMON-SUBSEQUENCE($X$, $Y$)

$m \leftarrow \text{length}(X)$;

$n \leftarrow \text{length}(Y)$;

// initialization
for $i \leftarrow 0$ to $m$
    $d[i, 0] \leftarrow 0$;
for $j \leftarrow 0$ to $n$
    $d[0, j] \leftarrow 0$;

// dynamic programming
for $i \leftarrow 1$ to $m$
    for $j \leftarrow 1$ to $n$
        if ($x_i = y_j$)
            $d[i, j] \leftarrow d[i - 1, j - 1] + 1$;
            $p[i, j] \leftarrow \text{"LU"}$; // "LU" indicates left up arrow
        else
if \((d[i - 1, j] \geq d[i, j - 1])\)
\[
d[i, j] \leftarrow d[i - 1, j];
\]
\[
p[i, j] \leftarrow "U"; \quad // "U" indicates up arrow
\]
else
\[
d[i, j] \leftarrow d[i, j - 1];
\]
\[
p[i, j] \leftarrow "L"; \quad // "L" indicates left
\]
end if
end if
end for
end for

return \((d, p)\);

Since it takes only \(O(1)\) time to fill in each of the \(O(mn)\) table entries the algorithm runs in \(O(mn)\) time.
Step 4: Construction of Optimal Solution

As mentioned before, we also maintain a $m \times n$ matrix $p$ for storing arrows to reconstruct the elements of the LCS. The following recursive procedure prints out an LCS of $X$ and $Y$.

$$\text{PRINT-LCS}(p, X, i, j)$$

- if $(i = 0 || j = 0)$ return NULL;
- if $(p[i, j] = \text{"LU"})$
  - PRINT-LCS($p, X, i - 1, j - 1$);
  - print $x_i$;
- else
  - if $(p[i, j] = \text{"U"})$
    - PRINT-LCS($p, X, i - 1, j$);
  - else PRINT-LCS($p, X, i, j - 1$);
end if
A slightly different problem with a similar solution

Given two strings $X = x_1x_2...x_m$ and $Y = y_1y_2...y_n$, find their longest common substring $Z$, i.e., a largest largest $k$ for which there are indices $i$ and $j$ with $x_ix_{i+1}...x_{i+k-1} = y_jy_{j+1}...y_{j+k-1}$.

For example:
$X :$ DEADBEEF
$Y :$ EATBEEF
$Z :$ BEEF ⁄ ⁄ pick the longest contiguous substring

Show how to do this in time $O(mn)$ by dynamic programming.
Step 1: Space of Subproblems

For $1 \leq i \leq m$, and $1 \leq j \leq n$,

- First Attempt: Define $d'_{i,j}$ to be the length of the longest common substring of $X[1..i]$ and $Y[1..j]$. (Does this work?)
- Second Attempt: Define $d_{i,j}$ to be the length of the longest common substring ending at $x_i$ and $y_j$. (Does this work?)
- Let $D$ be the $m \times n$ matrix $[d_{i,j}]$.
  - How does $D$ provide answer?
Step 2: Recursive Formulation

Case 1: If \( x_i = y_j \), then \( z_k = x_i = y_j \) and

\[ Z_{k-1} \text{ is a LCS of } X \text{ and } Y \text{ ending at } x_{i-1} \text{ and } y_{j-1} \]

Case 2: If \( x_i \neq y_j \), then there can’t be a common substring ending at \( x_i \) and \( y_j \)!

\[
d_{i,j} = \begin{cases} 
  d_{i-1,j-1} + 1 & \text{if } x_i = y_j \\
  0 & \text{if } x_i \neq y_j
\end{cases}
\]

Finally, we can find length of longest common substring by finding maximum \( d_{i,j} \) among all possible ending positions \( i \) and \( j \).

\[
\text{LCSString}(X, Y) = \max\{d_{i,j}\}
\]
Step 3: Bottom-up Computation

Similar to Longest Common Subsequence we set the first row and column of the matrix \( d[0,j] \) and \( d[i,0] \) to be 0.

Calculate \( d[1,j] \) for \( j = 1, 2, \ldots, n \).

Then, the \( d[2,j] \) for \( i = 1, 2, \ldots, 2, \)

Then, the \( d[3,j] \) for \( i = 1, 2, \ldots, 2, \)

etc., filling the matrix row by row and left to right.

For this problem we do not need to create another \( m \times n \) matrix for storing arrows. Instead, we use \( l_{\text{max}} \) and \( p_{\text{max}} \) to store the largest length of common substring and its \( i \) position respectively. This suffices to reconstruct the solution.
LONGEST-COMMON-SUBSTRING\((X, Y)\)

\[m \leftarrow \text{length}(X);\]
\[n \leftarrow \text{length}(Y);\]
\[l_{\text{max}} \leftarrow 0;\]
\[p_{\text{max}} \leftarrow 0;\]

// initialization

\[
\text{for } i \leftarrow 0 \text{ to } m
\]
\[
\hspace{1em} d[i, 0] \leftarrow 0;
\]
\[
\text{for } j \leftarrow 0 \text{ to } n
\]
\[
\hspace{1em} d[0, j] \leftarrow 0;
\]

// dynamic programming

\[
\text{for } i \leftarrow 1 \text{ to } m
\]
\[
\hspace{1em} \text{for } j \leftarrow 1 \text{ to } n
\]
\[
\hspace{2em} \text{if } (x_i \neq y_j)
\]
\[
\hspace{3em} d[i, j] \leftarrow 0;
\]
else
    \( d[i, j] \leftarrow d[i - 1, j - 1] + 1; \)
    \textbf{if} (\( d[i, j] > l_{\text{max}} \))
        \( l_{\text{max}} \leftarrow d[i, j]; \)
        \( p_{\text{max}} \leftarrow i; \)
    \textbf{end if}
\textbf{end if}
\textbf{end for}
\textbf{end for}

\textbf{return} \( l_{\text{max}}, p_{\text{max}}; \)

The dynamic programming algorithm runs in \( O(mn) \) time.
Step 4: Construction of Optimal Solution

Since we maintained \( l_{\text{max}} \) and \( p_{\text{max}} \), we can use them to print out the longest common substring of \( X \) and \( Y \) in the following procedure.

\[
\text{PRINT-LCSUBSTRING}(X, p_{\text{max}}, l_{\text{max}})
\]

\[
\begin{align*}
\text{if} \ (l_{\text{max}} = 0) & \quad \text{return} \ \text{NULL}; \\
\text{for} \ i & \leftarrow (p_{\text{max}} - l_{\text{max}} + 1) \ \text{to} \ p_{\text{max}} \\
\ & \quad \text{print} \ x_i;
\end{align*}
\]