Dynamic Programming

- An algorithm design technique similar to divide and conquer but unlike divide&conquer, subproblems may overlap in this case.

- Divide and conquer
  - Partition the problem into subproblems (may overlap)
  - Solve the subproblems recursively
  - Combine the solutions to solve the original problem

- Used for optimization problems
  - Goal: find an optimal solution (minimum or maximum)
  - There may be many solutions that lead to an optimal value
Dynamic Programming

- Applicable when subproblems are not independent
  - Subproblems share subsubproblems

  \[
  \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}
  \]

  **e.g.:** Combinations:

  - \( \binom{n}{1} = n \)
  - \( \binom{n}{n} = 1 \)

- Dynamic programming solves every subproblem and stores the answer in a table
Example: Combinations

\[
\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \\
\binom{n}{1} = n \\
\binom{n}{n} = 1
\]
Dynamic Programming Algorithm

1. **Characterize** the structure of an optimal solution

2. **Recursively** define the value of an optimal solution
   - An optimal solution to a problem contains within it an optimal solution to subproblems.
   - Typically, the recursion tree contains many overlapping subproblems.

3. **Compute** the value of an optimal solution in a bottom-up fashion
   - Optimal solution to the entire problem is build in a bottom-up manner from optimal solutions to subproblems.

4. **Construct** an optimal solution from computed information
Longest Common Subsequence

- Given two sequences
  \[ X = \langle x_1, x_2, \ldots, x_m \rangle \]
  \[ Y = \langle y_1, y_2, \ldots, y_n \rangle \]
  find a maximum length common subsequence (LCS) of \( X \) and \( Y \)

- *e.g.*: If \( X = \langle A, B, C, B, D, A, B \rangle \)
  Subsequences of \( X \):
  A subset of elements in the sequence taken in order
  \[ \langle A, B, D \rangle, \langle B, C, D, B \rangle, \langle B, C, D, A, B \rangle \text{ etc.} \]
Example

- \( \langle B, C, B, A \rangle \) and \( \langle B, D, A, B \rangle \) are longest common subsequences of \( X \) and \( Y \) \( (\text{length} = 4) \)

- \( \langle B, C, A \rangle \), however, is not a LCS of \( X \) and \( Y \)
Brute-Force Solution

- For every subsequence of X, check whether it’s a subsequence of Y

- There are $2^m$ subsequences of X to check

- Each subsequence takes $\Theta(n)$ time to check
  - scan Y for first letter, from there scan for second, and so on

- **Running time:** $\Theta(n2^m)$
Making the choice

\[ X = \langle A, B, D, G, E \rangle \]
\[ Y = \langle Z, B, D, E \rangle \]

- **Choice:** include one element into the common sequence (E) and solve the resulting subproblem

  \[ X = \langle A, B, D, G \rangle \]
  \[ Y = \langle Z, B, D \rangle \]

- **Choice:** exclude an element from a string and solve the resulting subproblem
Notations

- Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$
  we define the **i-th prefix** of $X$, for $i = 0, 1, 2, \ldots, m$

  $$X_i = \langle x_1, x_2, \ldots, x_i \rangle$$

- $c[i, j] = \text{the length of a LCS of the sequences}$
  $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$
A Recursive Solution

Case 1: $x_i = y_j$

e.g.: $X_i = \langle A, B, D, G, E \rangle$

$Y_j = \langle Z, B, D, E \rangle$

$c[i, j] = c[i - 1, j - 1] + 1$

- Append $x_i = y_j$ to the LCS of $X_{i-1}$ and $Y_{j-1}$

- Must find a LCS of $X_{i-1}$ and $Y_{j-1}$
A Recursive Solution

Case 2: \( x_i \neq y_j \)

* e.g.: \( X_i = \langle A, B, D, G \rangle \)
  \( Y_j = \langle Z, B, D \rangle \)

• Must solve two problems
  • find a LCS of \( X_{i-1} \) and \( Y_j \): \( X_{i-1} = \langle A, B, D \rangle \) and \( Y_j = \langle Z, B, D \rangle \)
  • find a LCS of \( X_i \) and \( Y_{j-1} \): \( X_i = \langle A, B, D, G \rangle \) and \( Y_{j-1} = \langle Z, B \rangle \)

\[
c[i, j] = \max \{ c[i - 1, j], c[i, j-1] \}
\]

• Optimal solution to a problem includes optimal solutions to subproblems
Overlapping Subproblems

- To find a LCS of \((X_m \text{ and } Y_n)\)
  - we may need to find
    - the LCS between \(X_m\) and \(Y_{n-1}\) and that of \(X_{m-1}\) and \(Y_n\)
  - Both of the above subproblems has the subproblem of finding the LCS of \((X_{m-1} \text{ and } Y_{n-1})\)

- Subproblems share subsubproblems
Computing the Length of the LCS

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

<table>
<thead>
<tr>
<th></th>
<th>Y_j:</th>
<th>Y_1</th>
<th>Y_2</th>
<th>Y_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x_i:</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x_1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x_2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>x_m</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First:
- \( x_1 \)
- \( x_2 \)

Second:
- \( x \_m \)

\( i \) is the index of the first set, \( j \) is the index of the second set.
A matrix $b[i, j]$:

- For a subproblem $[i, j]$ it tells us what choice was made to obtain the optimal value.
  - If $x_i = y_j$
    $$b[i, j] = "\downarrow"$$
  - Else, if $c[i - 1, j] \geq c[i, j - 1]$
    $$b[i, j] = "\uparrow"$$
  - Else
    $$b[i, j] = "\leftarrow"$$

A matrix $b[i, j]$:

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

\[
X = \langle A, B, C, B, D, A, B \rangle
\]
\[
Y = \langle B, D, C, A, B, A \rangle
\]

If \( x_i = y_j \),
\[
b[i, j] = " \downarrow \uparrow "
\]
else if \( c[i - 1, j] \geq c[i, j - 1] \),
\[
b[i, j] = " \uparrow \uparrow "
\]
else
\[
b[i, j] = " \leftarrow \rightarrow "
\]

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

- Start at \( b[m, n] \) and follow the arrows
- When we encounter a \( \Rightarrow \) in \( b[i, j] \) \( \Rightarrow x_i = y_j \) is an element of the LCS

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<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>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
LCS-LENGTH(X, Y, m, n)

1. for i ← 1 to m
2. do c[i, 0] ← 0
3. for j ← 0 to n
4. do c[0, j] ← 0
5. for i ← 1 to m
6. do for j ← 1 to n
7. do if x_i = y_j
8. then c[i, j] ← c[i - 1, j - 1] + 1
9. b[i, j] ← "\downarrow"
10. else if c[i - 1, j] ≥ c[i, j - 1]
11. then c[i, j] ← c[i - 1, j]
12. b[i, j] ← "↑"
13. else c[i, j] ← c[i, j - 1]
14. b[i, j] ← "←"
15. return c and b

If one of the sequences is empty, the length of the LCS is zero

Case 1: x_i = y_j
Case 2: x_i ≠ y_j

Running time : \( \Theta(mn) \)
PRINT-LCS(b, X, i, j)

1. if i = 0 or j = 0
2. then return
3. if b[i, j] = " \"
4. then PRINT-LCS(b, X, i - 1, j - 1)
5. print X_i
6. elseif b[i, j] = "↑"
7. then PRINT-LCS(b, X, i - 1, j)
8. else PRINT-LCS(b, X, i, j - 1)

Initial call: PRINT-LCS(b, X, length[X], length[Y])

Running time: \( \Theta(m + n) \)
Improving the Code

- What can we say about how each entry $c[i, j]$ is computed?
  - It depends only on $c[i - 1, j - 1]$, $c[i - 1, j]$, and $c[i, j - 1]$
  - Eliminate table $b$ and compute in $O(1)$ which of the three values was used to compute $c[i, j]$
  - We save $\Theta(mn)$ space from table $b$
  - However, we do not asymptotically decrease the auxiliary space requirements: still need table $c$

- If we only need the length of the LCS
  - LCS-LENGTH works only on two rows of $c$ at a time
    - The row being computed and the previous row
  - We can reduce the asymptotic space requirements by storing only these two rows