CS 2500: Algorithms

Lecture 25: Dynamic Programming: Longest Common Subsequence

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

November 12, 2024

Introduction to Longest Common Subsequence (LCS)

- Biological applications often compare DNA sequences of different organisms.
- A DNA strand is a string of bases represented by the letters: **A** (adenine), **C** (cytosine), **G** (guanine), and **T** (thymine).
- We can express DNA strands as sequences over the set $\{A, C, G, T\}$.
- Example DNA sequences:
  - $S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
  - $S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA
- Goal: Measure similarity between S_1 and S_2.

Measuring DNA Sequence Similarity

- Different approaches to measuring similarity:
  - Substring similarity: Check if one sequence is a substring of the other.
  - Edit distance: Number of changes needed to transform one sequence into another.
  - Common subsequence similarity: Find the longest sequence of bases appearing in both sequences in the same order.
- We focus on finding the longest common subsequence (LCS) as a similarity measure.

Example of LCS

- Consider sequences S_1 and S_2:
  - $S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
  - $S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA
- The longest common subsequence (LCS) is:

$$S_3 = \text{GTCGTCGGAAGCCGGCCGAA}$$

- The LCS gives a measure of similarity between two DNA sequences by finding a maximal length sequence common to both.

Subsequence

- A **subsequence** of a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$ is a sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ where:
  - There exists a strictly increasing sequence of indices $i_1, i_2, \ldots, i_k$ such that $x_{i_j} = z_j$ for all $j = 1, 2, \ldots, k$.
- Example: $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ with indices $2, 3, 5, 7$.

Common Subsequence

- Given two sequences X and Y, a sequence Z is a common subsequence if Z is a subsequence of both X and Y.
- Example:
  - $X = \langle A, B, C, B, D, A, B \rangle$
  - $Y = \langle B, D, C, A, B, A \rangle$
  - One common subsequence is $Z = \langle B, C, A \rangle$, but it is not the longest common subsequence.

Longest Common Subsequence

- In the LCS problem, given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, the goal is to find the longest sequence that is a subsequence of both X and Y.
- Example LCS:
  - For $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$
  - The LCS is $\langle B, C, B, A \rangle$.

Brute-Force Approach to LCS

- The brute-force approach for solving the LCS problem:
  - Enumerate all subsequences of X.
  - For each subsequence of X, check if it is also a subsequence of Y.
  - Keep track of the longest common subsequence found.
- This approach requires exponential time, $O(2^m)$, as there are 2^m subsequences of X.
- Impractical for long sequences due to the high time complexity.

Optimal Substructure Property of LCS

- The LCS problem has an **optimal-substructure property**.
- Optimal substructure means an optimal solution to the problem contains within it optimal solutions to subproblems.
- For LCS, natural subproblems correspond to pairs of prefixes of the input sequences.
- Define the i-th prefix of $X = \langle x_1, x_2, \ldots, x_m \rangle$ as:

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

- Example: If $X = \langle A, B, C, B, D, A, B \rangle$, then $X_4 = \langle A, B, C, B \rangle$.

Theorem: Optimal Substructure of an LCS

- Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences.
- Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.
- Theorem:
  1. If the last characters of X and Y are the same (i.e., $x_m = y_n$), then the last character of Z must also be x_m, and the rest of Z (denoted Z_{k-1}) is an LCS of X_{m-1} and Y_{n-1}, which are X and Y without their last characters.
  2. If the last characters of X and Y are different (i.e., $x_m \neq y_n$) and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
  3. Similarly, if $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of X and Y_{n-1}.

Case 1: $x_m = y_n$

If the last characters of X and Y are the same, i.e., $x_m = y_n$, then:

- We claim that the last character of Z (which is an LCS of X and Y) must be $x_m = y_n$, so $z_k = x_m = y_n$.
- If this were not true (i.e., if $z_k \neq x_m = y_n$), we could add x_m to Z, forming a common subsequence of X and Y with a length of $k + 1$. This contradicts the assumption that Z is a longest common subsequence.

Therefore, $z_k = x_m = y_n$, and the remainder of Z (denoted Z_{k-1}) is an LCS of X_{m-1} and Y_{n-1}.

Proof of Theorem

Case 1: Example

Let:

- $X = \langle A, B, C, D, E \rangle$
- $Y = \langle C, B, D, E \rangle$

If the LCS is $Z = \langle C, D, E \rangle$, observe that:

- The last characters of X and Y are both E (i.e., $x_5 = y_4 = E$).
- Since Z also ends with E, the remaining subsequence $Z_{k-1} = \langle C, D \rangle$ must be an LCS of the prefixes $X_4 = \langle A, B, C, D \rangle$ and $Y_3 = \langle C, B, D \rangle$.

Proof of Theorem

Case 2: $x_m \neq y_n$ **and** $z_k \neq x_m$
If $x_m \neq y_n$ and the last character of Z (i.e., z_k) is not equal to x_m, then:

- Z must be a longest common subsequence of X_{m-1} and Y.

If this were not the case (i.e., if there were a common subsequence W of X_{m-1} and Y with a length greater than k), then W would also be a common subsequence of X and Y, contradicting the assumption that Z is an LCS of X and Y.

Proof of Theorem

Case 2: Example

Let:

- $X = \langle A, B, C, D, F \rangle$
- $Y = \langle B, C, D, E \rangle$

Suppose the LCS is $Z = \langle B, C, D \rangle$.

- The last characters of X and Y are F and E, so $x_5 \neq y_4$.
- Since Z does not contain x_5 or y_4, Z must be an LCS of either:
  - $X_{m-1} = \langle A, B, C, D \rangle$ and $Y = \langle B, C, D, E \rangle$, or
  - $X = \langle A, B, C, D, F \rangle$ and $Y_{n-1} = \langle B, C, D \rangle$.
- Thus, $Z = \langle B, C, D \rangle$ is an LCS of both cases.

Case 3: $x_m \neq y_n$ **and** $z_k \neq y_n$
If $x_m \neq y_n$ and $z_k \neq y_n$, then:

- Z must be a longest common subsequence of X and Y_{n-1}.

If this were not the case (i.e., if there were a common subsequence W of X and Y_{n-1} with a length greater than k), then W would also be a common subsequence of X and Y, contradicting the assumption that Z is an LCS of X and Y.

Case 3: Example

Let:

- $X = \langle A, B, C, D \rangle$
- $Y = \langle B, C, E, F \rangle$

Suppose the LCS is $Z = \langle B, C \rangle$.

- The last characters of X and Y are D and F, so $x_4 \neq y_4$.
- Since Z does not contain x_4 or y_4, Z must be an LCS of either:
  - $X_{n-1} = \langle A, B, C \rangle$ and $Y = \langle B, C, E, F \rangle$, or
  - $X = \langle A, B, C, D \rangle$ and $Y_{n-1} = \langle B, C, E \rangle$.
- Thus, $Z = \langle B, C \rangle$ is an LCS of both cases.

Significance of Theorem

- The theorem characterizes the structure of LCS.
- Shows that any LCS of two sequences contains within it an LCS of prefixes of the sequences.
- This recursive structure underpins the dynamic programming approach to solving the LCS problem.

LCS Problem Setup and Notation

Problem: Given two sequences:

- $X = \langle x_1, x_2, \ldots, x_m \rangle$
- $Y = \langle y_1, y_2, \ldots, y_n \rangle$

Our goal is to find the longest common subsequence (LCS) of X and Y.

Optimal Substructure: The theorem tells us that to solve the LCS problem, we need to examine specific subproblems:

- If the last characters of X and Y match ($x_m = y_n$), then they must be part of the LCS.
- If the last characters do not match ($x_m \neq y_n$), we consider two possibilities to find the longer subsequence.

LCS: Top-Down Approach with Memoization

Base Case:

- The LCS problem requires us to consider the lengths of the subsequences as we recurse.
- If either X or Y has length 0 (i.e., $m = 0$ or $n = 0$), then no common subsequence exists.
- Therefore, the base case for the recursive function is:

$$\text{LCS}(X_m, Y_n) = 0 \quad \text{if } m = 0 \text{ or } n = 0$$

- This base case stops the recursion when we reach the end of either sequence.

LCS: Top-Down Approach with Memoization

Recursive Case: Matching Last Characters

- When the last characters of X and Y match ($x_m = y_n$):
  - The character $x_m = y_n$ is part of the LCS.
  - We can reduce both sequences by one character to solve the subproblem for the remaining prefixes X_{m-1} and Y_{n-1}.
- Recursive function for this case:

$$\text{LCS}(X_m, Y_n) = \text{LCS}(X_{m-1}, Y_{n-1}) + 1 \quad \text{if } x_m = y_n$$

- This case follows from the optimal substructure property (Theorem, Case 1).

LCS: Top-Down Approach with Memoization

Recursive Case: Non-Matching Last Characters

- When the last characters of X and Y do not match ($x_m \neq y_n$):
  - We cannot include x_m or y_n in the LCS directly, but we need to consider two possible subproblems to find the longest subsequence.
- The recursive cases:
  1. Compute the LCS of X_{m-1} and Y (ignoring the last character of X):
  $$\mathrm{LCS}(X_{m-1}, Y_n)$$
  2. Compute the LCS of X and Y_{n-1} (ignoring the last character of Y):
  $$\mathrm{LCS}(X_m, Y_{n-1})$$
- Since we want the longest common subsequence, we take the maximum of these two values:

$$\mathrm{LCS}(X_m, Y_n) = \max(\mathrm{LCS}(X_{m-1}, Y_n), \mathrm{LCS}(X_m, Y_{n-1})) \quad \text{if } x_m \neq y_n$$

Recursive Formula for LCS:

- Combining both cases, we obtain the recursive formula:

$$\mathrm{LCS}(X_m, Y_n) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ \mathrm{LCS}(X_{m-1}, Y_{n-1}) + 1 & \text{if } m > 0, n > 0, \text{ and } x_m = y_n \\ \max(\mathrm{LCS}(X_{m-1}, Y_n), \mathrm{LCS}(X_m, Y_{n-1})) & \text{if } m > 0, n > 0, \text{ and } x_m \neq y_n \end{cases}$$

- This formula defines the length of an LCS of the prefixes X_m and Y_n of sequences X and Y.

LCS: Top-Down Approach with Memoization

Example:

- Let $X = \langle \text{a, b, c, d, g, h} \rangle$ and $Y = \langle \text{a, b, e, d, f, h, r} \rangle$.
- Start with the last characters:
  - $x_6 = h$ and $y_7 = r$, which are not equal.
- Apply the recursive cases:
  1. Calculate $\text{LCS}(X_{m-1}, Y_n) = \text{LCS}(X[1 \ldots 5], Y[1 \ldots 7])$.
  2. Calculate $\text{LCS}(X_m, Y_{n-1}) = \text{LCS}(X[1 \ldots 6], Y[1 \ldots 6])$.
- Continue until reaching the base case.
- The final answer is the length of the longest LCS found.

LCS: Top-Down Approach with Memoization

Algorithm 1 LCS - Top-Down with Memoization

Require: X: string of length m, Y: string of length n
Ensure: Length of LCS of X and Y

Initialize a memoization table $L[0 \dots m][0 \dots n]$ with all values set to -1
function LCS(i, j)
  if $i = 0$ or $j = 0$ **then**
    return 0
  end if
  if $L[i][j] \neq -1$ **then**
    return $L[i][j]$
  end if
  if $x_i = y_j$ **then**
    $L[i][j] \leftarrow \text{LCS}(i-1, j-1) + 1$
  else
    $L[i][j] \leftarrow \max(\text{LCS}(i-1, j), \text{LCS}(i, j-1))$
  end if
  return $L[i][j]$
end function
Compute the solution: Call LCS(m, n)

LCS: Bottom-Up Approach

Bottom-Up Table Construction:

- Define a 2D table $L[0 \ldots m][0 \ldots n]$ where $L[i][j]$ represents the length of the LCS of $X[1 \ldots i]$ and $Y[1 \ldots j]$.
- Base cases:
  - $L[0][j] = 0$ for all j (LCS with an empty string is 0).
  - $L[i][0] = 0$ for all i.
- Recurrence relation:

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

- Final result: $L[m][n]$ contains the length of the LCS of X and Y.

LCS: Bottom-Up Approach

Algorithm 2 LCS - Bottom-Up Approach

Require: X: string of length m, Y: string of length n
Ensure: Length of LCS of X and Y

```
Initialize a table L[0...m][0...n] with all values set to 0
for i = 1 to m do
    for j = 1 to n do
        if x_i = y_j then
            L[i][j] ← L[i−1][j−1] + 1
        else
            L[i][j] ← max(L[i−1][j], L[i][j−1])
        end if
    end for
end for
return L[m][n]                          ▷ Length of the LCS of X and Y
```

LCS: Bottom-Up Approach

Example:

- $X = \langle A, B, C, D \rangle$
- $Y = \langle B, D, C, A \rangle$

- Fill the table $L[i][j]$ step-by-step, using the recurrence relation.
- Use base cases and recurrence to compute each entry.
- The final result at $L[m][n]$ will provide the length of the LCS.

LCS: Bottom-Up Approach–Filling the DP Table for LCS

Goal: Fill each cell $dp[i][j]$ to find the longest common subsequence (LCS) of X and Y.

- Each cell $dp[i][j]$ represents the LCS length of prefixes $X[0:i]$ and $Y[0:j]$.
- **Check three neighboring cells:**
  - **Diagonal (Top-Left Neighbor):** $dp[i-1][j-1]$
    - Use if $X[i-1] = Y[j-1]$, indicating a match.
    - Set $dp[i][j] = dp[i-1][j-1] + 1$.
  - **Left Neighbor:** $dp[i][j-1]$
    - Use if $X[i-1] \neq Y[j-1]$. Take the max to carry forward the LCS length.
  - **Top Neighbor:** $dp[i-1][j]$
    - Use if $X[i-1] \neq Y[j-1]$. Take the max to carry forward the LCS length.

LCS: Bottom-Up Approach–Filling the DP Table for LCS

Goal: Fill each cell $dp[i][j]$ to find the longest common subsequence (LCS) of X and Y.

- **Mnemonic:**
  - If characters match: Use Diagonal cell $+1$.
  - If characters don't match: Take the max of Left and Top neighbors.
- Final value at $dp[m][n]$ gives the length of the LCS of X and Y.

Example Cell Filling

- **Match?** Diagonal cell $+1$
- **No match?** Max of Left and Top

Initial Table:

- Initialize the table L with all values set to 0.
- Use the rule:
  - **Match?** Diagonal $+1$
  - **No Match?** Max of Top and Left

$L[i][j]$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	0	0	0	0	0
$i = 1$	0				
$i = 2$	0				
$i = 3$	0				
$i = 4$	0				

LCS: Bottom-Up Approach

Filling Row $i = 1$ (Comparing $x_1 = A$)

$L[i][j]$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	0	0	0	0	0
$i = 1$	0	0	0	0	1
$i = 2$	0				
$i = 3$	0				
$i = 4$	0				

- $j = 1$: $x_1 = A$ and $y_1 = B \neq A \Rightarrow$ Max of Top and Left $= 0$.
- $j = 2$: $x_1 = A$ and $y_2 = D \neq A \Rightarrow$ Max of Top and Left $= 0$.
- $j = 3$: $x_1 = A$ and $y_3 = C \neq A \Rightarrow$ Max of Top and Left $= 0$.
- $j = 4$: $x_1 = A$ and $y_4 = A \Rightarrow$ Diagonal $+ 1 = 1$.

LCS: Bottom-Up Approach

Filling Row $i = 2$ (Comparing $x_2 = B$)

$L[i][j]$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	0	0	0	0	0
$i = 1$	0	0	0	0	1
$i = 2$	0	1	1	1	1
$i = 3$	0				
$i = 4$	0				

- $j = 1$: $x_2 = B$ and $y_1 = B \Rightarrow$ Diagonal $+ 1 = 1$.
- $j = 2$: $x_2 = B$ and $y_2 = D \neq B \Rightarrow$ Max of Top and Left $= 1$.
- $j = 3$: $x_2 = B$ and $y_3 = C \neq B \Rightarrow$ Max of Top and Left $= 1$.
- $j = 4$: $x_2 = B$ and $y_4 = A \neq B \Rightarrow$ Max of Top and Left $= 1$.

LCS: Bottom-Up Approach

Filling Row $i = 3$ (Comparing $x_3 = C$)

$L[i][j]$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	0	0	0	0	0
$i = 1$	0	0	0	0	1
$i = 2$	0	1	1	1	1
$i = 3$	0	1	1	2	2
$i = 4$	0				

- $j = 1$: $x_3 = C$ and $y_1 = B \neq C \Rightarrow$ Max of Top and Left $= 1$.
- $j = 2$: $x_3 = C$ and $y_2 = D \neq C \Rightarrow$ Max of Top and Left $= 1$.
- $j = 3$: $x_3 = C$ and $y_3 = C \Rightarrow$ Diagonal $+ 1 = 2$.
- $j = 4$: $x_3 = C$ and $y_4 = A \neq C \Rightarrow$ Max of Top and Left $= 2$.

LCS: Bottom-Up Approach

Filling Row $i = 4$ (Comparing $x_4 = D$)

$L[i][j]$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 0$	0	0	0	0	0
$i = 1$	0	0	0	0	1
$i = 2$	0	1	1	1	1
$i = 3$	0	1	1	2	2
$i = 4$	0	1	2	2	2

- $j = 1$: $x_4 = D$ and $y_1 = B \neq D \Rightarrow$ Max of Top and Left $= 1$.
- $j = 2$: $x_4 = D$ and $y_2 = D \Rightarrow$ Diagonal $+ 1 = 2$.
- $j = 3$: $x_4 = D$ and $y_3 = C \neq D \Rightarrow$ Max of Top and Left $= 2$.
- $j = 4$: $x_4 = D$ and $y_4 = A \neq D \Rightarrow$ Max of Top and Left $= 2$.

LCS: Summary

Key Points:

- The LCS problem can be solved using both Top-Down and Bottom-Up dynamic programming approaches.
- Top-Down uses recursion and memoization, while Bottom-Up fills the table iteratively.
- Both approaches achieve $O(m \cdot n)$ time complexity.
- The Bottom-Up approach is often preferred due to lower recursion overhead.

Problem: Longest Common Substring (LCSstr)

Problem Definition:

- Given two strings:
  - $X = \langle x_1, x_2, \ldots, x_m \rangle$
  - $Y = \langle y_1, y_2, \ldots, y_n \rangle$
- Find the longest contiguous substring that appears in both X and Y.

Example:

- For $X =$ "ABABC" and $Y =$ "BABCA", the longest common substring is **"BABC"**, with length 4.

Comparing Recurrence Relations for LCS and LCSstr

- **LCS Recurrence Relation:**
  - If $X[i] = Y[j]$:
  $$L[i][j] = L[i-1][j-1] + 1$$
  - If $X[i] \neq Y[j]$:
  $$L[i][j] = \max(L[i-1][j], L[i][j-1])$$
  - This allows non-contiguous subsequences by taking the maximum of subproblems without resetting the count.
- **LCSstr Recurrence Relation:**
  - If $X[i] = Y[j]$:
  $$L[i][j] = L[i-1][j-1] + 1$$
  - If $X[i] \neq Y[j]$:
  $$L[i][j] = 0$$
  - For LCSstr, contiguity is required, so any mismatch resets the count to 0, unlike LCS.

Using LCS to Derive LCSstr Recurrence

- The LCS recurrence gives a framework to compare elements $X[i]$ and $Y[j]$.
- For LCSstr, we adapt this by resetting $L[i][j] = 0$ on mismatches to enforce contiguity.
- This modification creates a recurrence tailored for contiguous substrings.

LCSstr: Base Case and Algorithm Outline

Base Case:

- $L[i][0] = 0$ and $L[0][j] = 0$ for all i and j.
- If either string has length 0, the longest common substring length is 0.

Algorithm Outline (Bottom-Up):

- Initialize a 2D table L with all values set to 0.
- For each i from 1 to m:
  - For each j from 1 to n:
    - If $X[i-1] = Y[j-1]$: $L[i][j] = L[i-1][j-1] + 1$.
    - Update ‘maxLength‘ if $L[i][j]$ exceeds the current ‘maxLength‘.
    - Otherwise, set $L[i][j] = 0$.
- ‘maxLength‘ stores the length of the longest common substring.

Pseudo-Code for LCSstr (Bottom-Up Approach)

Algorithm 1 LCSstr - Bottom-Up Approach

Require: X: string of length m, Y: string of length n

Ensure: Length of the longest common substring

```
Initialize a table L[0...m][0...n] with all values set to 0
maxLength ← 0
for i = 1 to m do
    for j = 1 to n do
        if X[i − 1] = Y[j − 1] then
            L[i][j] ← L[i − 1][j − 1] + 1
            maxLength = max(maxLength, L[i][j])
        else
            L[i][j] ← 0
        end if
    end for
end for
return maxLength
```

LCSstr: Example

Example: X = "ABABC" and Y = "BABCA"

$L[i][j]$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=0$	0	0	0	0	0	0
$i=1$	0	0	1	0	0	1
$i=2$	0	1	0	2	0	0
$i=3$	0	0	2	0	0	0
$i=4$	0	1	0	3	0	0
$i=5$	0	0	0	0	4	0

Result: The longest common substring is **"BABC"** with length 4.

LCSstr: Complexity

Complexity:

- **Time Complexity:** $O(m \times n)$, where m and n are the lengths of X and Y.
- **Space Complexity:** $O(m \times n)$.

Problem: Minimum Insertions and Deletions

Goal: Given two strings A and B, find the minimum number of insertions and deletions required to transform A into B.

Example:

- $A =$ "heap"
- $B =$ "pea"

Solution:

- Convert A to B with minimum operations.
- Output: Number of deletions and insertions needed.

Key Idea: Using Longest Common Subsequence (LCS)

Observation:

- The Longest Common Subsequence (LCS) of A and B represents the longest sequence that can remain unchanged in both strings.
- Any character in A that is not part of the LCS must be deleted.
- Any character in B that is not part of the LCS must be inserted.

Key Idea: Using Longest Common Subsequence (LCS)

Transforming A to B:

- Let `LCS_length` be the length of the LCS of A and B.
- Then:
  - **Deletions** = `A_length - LCS_length`
  - **Insertions** = `B_length - LCS_length`

Example:

- $A =$ "heap", $B =$ "pea"
- LCS_length $= 2$ (Longest Common Subsequence: "ea")
- **Deletions** $= 4 - 2 = 2$
- **Insertions** $= 3 - 2 = 1$
- **Total Operations** $= 2$ deletions $+ 1$ insertion $= 3$

Algorithm Pseudo-Code

Algorithm 2 Minimum Insertions and Deletions to Convert A to B

Require: A, B: input strings

Ensure: Minimum number of insertions and deletions to convert A to B

1: Compute the length of LCS, LCS_length, using DP
2: **Deletions** $=$ len(A) $-$ LCS_length
3: **Insertions** $=$ len(B) $-$ LCS_length
4: **return** (Deletions, Insertions)

Problem: Longest Palindromic Subsequence (LPS)

Goal: Given a string X, find the length of the longest subsequence of X that is a palindrome.

Example:

- For $X =$ "BBABCBCAB", the longest palindromic subsequence is "BABCBAB" with length 7.

Using LCS to Solve LPS:

- We can leverage the Longest Common Subsequence (LCS) concept to find the LPS.
- This approach simplifies LPS by transforming it into an LCS problem.

Key Idea: Using LCS on the Reversed String

Steps:

- Let X be the original string, and X_{rev} be its reverse.
- Find the LCS of X and X_{rev}.
- The LCS between X and X_{rev} will be the longest palindromic subsequence.

Why This Works:

- A palindrome reads the same forward and backward.
- Thus, the longest sequence common to both X and its reverse X_{rev} must be palindromic.

Algorithm for LPS Using LCS

Algorithm 1 Longest Palindromic Subsequence via LCS

Require: X: input string of length m

Ensure: Length of the longest palindromic subsequence in X

Let X_{rev} be the reverse of X
Initialize a 2D array $L[0 \dots m][0 \dots m]$ for LCS computation
for $i = 0$ to m **do**
  for $j = 0$ to m **do**
    if $i == 0$ or $j == 0$ **then**
      $L[i][j] \leftarrow 0$ ▷ Base case: empty strings
    else if $X[i-1] == X_{\text{rev}}[j-1]$ **then**
      $L[i][j] \leftarrow L[i-1][j-1] + 1$
    else
      $L[i][j] \leftarrow \max(L[i-1][j], L[i][j-1])$
    end if
  end for
end for
return $L[m][m]$ ▷ Length of the longest palindromic subsequence

Example: LPS Using LCS

Example: X = "BBABCBCAB"

- Reverse X: X_{rev} = "BACBCBABB"
- Compute LCS of X and X_{rev}.
- The LCS length gives the length of the LPS, which is 7.
- The longest palindromic subsequence is "BABCBAB".

LPS: Complexity

Time Complexity: $O(m^2)$, where m is the length of the string X.
Space Complexity: $O(m^2)$, due to the 2D DP table.