Longest Palindromic Subsequence Documentation

📑 Table of Contents

1. 📄 Problem Statement

Given a string s, find the length of the longest subsequence in it that is a palindrome.

- A subsequence is a sequence derived from another sequence by deleting some or no elements without changing the order of the remaining elements.
- A palindrome reads the same forwards and backwards.

Constraints:

- 1 <= s.length <= 1000
- s consists only of lowercase English letters.

2. 💡 Intuition

Palindromic subsequences may skip characters in between, unlike substrings. So brute-forcing every subsequence and checking if it's a palindrome would be inefficient.

We need an efficient way to:

- Explore all possible subsequences.
- Store overlapping subproblems.
- Maximize the length of palindromic sequences.

3. 🔍 Key Observations

- A single character is always a palindrome of length 1.
- If s[i] == s[j], the outer characters can be part of a palindromic subsequence.
- If not equal, the result depends on the maximum between the subsequences excluding either s[i] or s[j].

4. 🔧 Approach

Use Dynamic Programming with a 2D table dp[i][j]:

- dp[i][j] = length of longest palindromic subsequence in s[i..j].

Transition:

- If s[i] == s[j]:
  dp[i][j] = 2 + dp[i+1][j-1]
- Else:
  dp[i][j] = max(dp[i+1][j], dp[i][j-1])

Initialization:

- All substrings of length 1 (i == j) are palindromes: dp[i][i] = 1

Fill Order:

- Start from length 2 up to n
- Move left to right for each length

5. ⚠ Edge Cases

- Empty string → Not applicable due to constraint s.length >= 1
- All characters are the same → Full string is a palindrome
- No repeated characters → Result is 1 (any single character)

6. 📊 Complexity Analysis

□ Time Complexity:

- O(n²) — where n is the length of the string.
- We compute dp[i][j] for all i ≤ j.

💾 Space Complexity:

- O(n²) — due to the 2D DP table.

7. 🔄 Alternative Approaches

- Recursion + Memoization: Top-down with caching, similar performance but uses call stack.
- Space Optimization: Reduce 2D to 1D by storing only previous row (complex indexing).
- Longest Common Subsequence (LCS): Reverse the string and find LCS between s and reverse(s).

8. □ Test Cases

Input	Expected Output	Explanation
"bbbab"	4	"bbbb" is a valid subsequence.
"cbbd"	2	"bb" is the longest palindrome.
"abcd"	1	No repeating characters.
"aaaa"	4	Whole string is a palindrome.
"a"	1	Single character string.

9. 🏁 Final Thoughts

- This problem is a great use case for dynamic programming.
- It teaches how to build subproblems for non-contiguous sequences.
- Optimizations can reduce memory, but the basic approach is already efficient for most constraints.