📘 Documentation: Longest Uncommon Subsequence I

1. Problem Statement

Given two strings a and b, return the length of the longest uncommon subsequence between them. If no such uncommon subsequence exists, return -1.

A subsequence is a string that can be derived from another string by deleting some characters without changing the order of the remaining characters.An uncommon subsequence is a subsequence that appears in one string but not the other.

2. Intuition

If two strings are not equal, the longer string itself is a valid uncommon subsequence, as it cannot be a subsequence of the shorter (or different) one.However, if both strings are exactly the same, then any subsequence of one is also a subsequence of the other—hence, no uncommon subsequence.

3. Key Observations

- If a == b, then every subsequence is shared → return -1.
- If a != b, then the longer string itself is a unique subsequence → return max(len(a), len(b)).

4. Approach

- Compare the strings:
  - If they are equal → return -1.
  - If they are not equal → return the length of the longer string.
- No need to explore all subsequences (which would be exponential time) due to the above insight.

5. Edge Cases

- a = "abc", b = "abc" → both are same → return -1.
- a = "a", b = "b" → both are different → return 1.
- a = "abcd", b = "ab" → different → return 4.

6. Complexity Analysis

🕒 Time Complexity

- O(1): Only comparison and length calculation.

□ Space Complexity

- O(1): No extra space used.

7. Alternative Approaches

- Brute-force: Generate all subsequences of both strings and check which ones are uncommon.
  - Not practical: Generating all subsequences has time complexity of O(2^n).
  - Our current solution is optimal for this problem's constraints.

8. Test Cases

a	b	Output	Explanation
"aba"	"cdc"	3	Both are different; each of length 3
"aaa"	"aaa"	-1	Both are identical
"abc"	"def"	3	Different strings; any one is a valid uncommon subsequence
"a"	"b"	1	Single-character strings, different
"abcd"	"abc"	4	"abcd" is not a subsequence of "abc"

9. Final Thoughts

This problem appears simple, but it's a great example of how recognizing patterns can avoid unnecessary computation.

Instead of diving into complex subsequence generation, the key is understanding how identity and length determine uniqueness.