521. Longest Uncommon Subsequence I

Easy String

Problem Description

The task is to find the length of the longest uncommon subsequence between two given strings `a` and `b`. An **uncommon subsequence** is defined as a sequence that can be found as a subsequence in one string but not in the other string.

A **subsequence** of a string is a series of characters that can be derived from the original string by deleting some or no characters without changing the order of the remaining characters. For instance, `"abc"` is a subsequence of `"aebdc"` because you can remove the characters `e` and `d` from `"aebdc"` to get `"abc"`.

The problem asks for the longest sequence that is a subsequence of one string and absolutely not a subsequence of the other string. If no such uncommon subsequence exists between `a` and `b`, the function should return `-1`. The uniqueness of the subsequence is important here; it must be unique to one string only.

Intuition

When looking for the longest uncommon subsequence between the two strings `a` and `b`, it's crucial to note that if the strings are identical, there can't be an uncommon subsequence, and we should return `-1`. This is because any subsequence of `a` would also be a subsequence of `b` and vice versa since they are the same.

Now, when the strings are different, the longest uncommon subsequence is actually the longer of the two strings. This is intuitive because a full string cannot be a subsequence of another shorter string. This means if `a` and `b` are different, the longest string itself can be considered the longest uncommon subsequence. This is why the provided solution returns the maximum of the lengths of `a` and `b` when `a` does not equal `b`.

Solution Approach

The solution involves a very simple approach and does not necessitate complex algorithms or data structures. It leverages fundamental properties of strings and subsequences.

Since a string is always a subsequence of itself and we're looking for the longest uncommon subsequence, the implementation checks for the following conditions:

- If `a` is equal to `b`, then every subsequence of `a` is also a subsequence of `b` and vice versa. Thus, there cannot be an uncommon subsequence, and the function returns `-1`.
- If `a` is not equal to `b`, the longest uncommon subsequence can be the longer string itself because the whole string cannot be a subsequence of the shorter string. This guarantees that the longest subsequence is unique to the longer string.

The code snippet provided in the reference solution can be explained in two parts:

```
if a == b:
    return -1
else:
    return max(len(a), len(b))
```

However, this can be simplified into a single line as it is in the solution:

```
return -1 if a == b else max(len(a), len(b))
```

The code includes a conditional expression that returns `-1` when `a` and `b` are identical (ensuring no uncommon subsequences exist) or it returns the length of the longer of the two strings.

No special patterns or algorithms are needed here since it boils down to a simple comparison and a straightforward application of the definition of a subsequence in the context of string comparison.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we have two strings `a = "horse"` and `b = "corpus"`. We want to determine the length of the longest uncommon subsequence between the two strings.

Here are the steps to solve this:

1. First, we compare the strings `a` and `b` to check if they are equal.
2. Since `"horse"` is not equal to `"corpus"`, we proceed to the next condition.
3. According to our approach, the longer of the two strings can be considered the longest uncommon subsequence because one cannot be a subsequence of the other.
4. We then get the lengths of `a` and `b`. For `a = "horse"`, the length is `5`. For `b = "corpus"`, the length is `6`.
5. The longest string of the two is `"corpus"` with a length of `6`.
6. Therefore, the length of the longest uncommon subsequence between `"horse"` and `"corpus"` is `6`.

In summary, by following the solution approach, the length of the longest uncommon subsequence between `a` and `b` is determined by simply comparing the lengths of `a` and `b` since they are not identical, resulting in a return value of `6` for this specific example.

Solution Implementation

Python

```python
class Solution:
    def findLUSlength(self, str_a: str, str_b: str) -> int:
        # Check if the two strings are identical
        if str_a == str_b:
            # The longest uncommon subsequence does not exist if strings are the same,
            # hence return -1 as specified by the problem
            return -1
        else:
            # If strings are not the same, return the length of the longer string
            # since the longer string itself will be the longest uncommon subsequence
            return max(len(str_a), len(str_b))
```

Java

```java
public class Solution {
    // Method to find the length of the Longest Uncommon Subsequence (LUS) between two strings
    public int findLUSlength(String firstString, String secondString) {
        // Check if both strings are equal
        if (firstString.equals(secondString)) {
            // If both strings are the same, there is no uncommon subsequence
            return -1;
        } else {
            // If the strings are not equal, the LUS is the length of the longer string
            // because the entire string itself is an uncommon subsequence
            return Math.max(firstString.length(), secondString.length());
        }
    }
}
```

C++

```cpp
class Solution {
public:
    int findLUSlength(string strA, string strB) {
        // If both strings are equal, there will be no uncommon subsequence.
        if (strA == strB) {
            return -1;
        }
        // If strings are not equal, the longest uncommon subsequence is the length
        // of the longer string because the two strings can't be subsequences of each other.
        return max(strA.size(), strB.size());
    }
};
```

TypeScript

```typescript
// This function finds the length of the Longest Uncommon Subsequence between two strings.
// If the strings are identical, it returns -1, indicating there is no uncommon subsequence.
// If the strings are different, it returns the length of the longer string.
//
// Parameters:
// a - The first string to compare.
// b - The second string to compare.
//
// Returns:
// The length of the longest uncommon subsequence, or -1 if no such subsequence exists.
function findLUSlength(a: string, b: string): number {
    // Compare the two strings; if they are not equal, return the length of the longer string.
    // If they are equal, return -1 as there are no uncommon subsequences.
    return a !== b ? Math.max(a.length, b.length) : -1;
}
```

```python
class Solution:
    def findLUSlength(self, str_a: str, str_b: str) -> int:
        # Check if the two strings are identical
        if str_a == str_b:
            # The longest uncommon subsequence does not exist if strings are the same,
            # hence return -1 as specified by the problem
            return -1
        else:
            # If strings are not the same, return the length of the longer string
            # since the longer string itself will be the longest uncommon subsequence
            return max(len(str_a), len(str_b))
```

Time and Space Complexity

The time complexity of the code is `O(1)`, because it performs a constant number of operations regardless of the input size. It only compares the two strings for equality and then finds the length of the longer string if they are not equal.

The space complexity of the code is also `O(1)` because it does not allocate any additional space that is dependent on the input size. The space used is for the input strings `a` and `b`, which is not counted towards space complexity as it is considered input space, and the space for storing the return value.