Documentation

The "Is Subsequence" problem is a string processing challenge where, given two strings s and t, the task is to determine if s is a subsequence of t. A subsequence is formed by deleting zero or more characters from a string while keeping the relative order of the remaining characters intact. In simpler terms, for s to be a subsequence of t, all characters of s should appear in t in the same order but not necessarily consecutively.

Problem Statement

The problem asks whether string s is a subsequence of string t. It is given that both strings consist only of lowercase English letters. The task is to return true if s is a subsequence of t or false otherwise. The problem also includes constraints where the length of s can be up to 100 characters and t can have a maximum length of 10^4. Additionally, a follow-up question asks how to optimize the solution when there are multiple queries to check whether different strings s are subsequences of t.

Intuition

The intuition behind solving this problem is simple: we need to check if we can iterate through the string t and find all characters of string s in the correct order. One straightforward approach is to use the two-pointer technique. This method works by using two pointers, one for each string, and iterating through t while trying to match the characters of s in order.

For the follow-up scenario with multiple s values, an optimized approach involves preprocessing string t to allow efficient lookup for each query. By storing the positions of each character in t, subsequent queries can be answered faster using binary search.

Key Observations

There are a few key observations that simplify the problem. First, if s is an empty string, it is always considered a subsequence of any string t. Second, if the length of s is greater than the length of t, s can't be a subsequence of t. Additionally, the characters in s must appear in t in the same order. The two-pointer approach is effective for most cases, but if we need to handle many queries, a preprocessing step can be applied to speed up the process.

Approach

The solution can be broken down into two main methods: the two-pointer approach and the binary search with a preprocessing approach.

1. **Two-Pointer Approach**: This is a simple and efficient method for solving the problem when checking a single subsequence. The idea is to use two pointers, i and j, initialized to 0, to traverse strings s and t respectively. We increment i whenever the characters at s[i] and t[j] match, and we always increment j to move through t. The process continues until either all characters of s are matched (in which case I will reach the end of s) or t is exhausted (in which case we return false). This approach has a time complexity of O(n), where n is the length of t.
2. **Binary Search with Preprocessing**: This method is more efficient when multiple subsequence queries need to be processed. In this approach, we first preprocess string t by storing the positions of all characters in a dictionary. For each query string s, we then use binary search to determine whether the characters of s appear in the correct order in t. The time complexity of this approach is O(m + k log m), where m is the length of t, and k is the length of s.

Edge Cases

There are several edge cases to consider in this problem. If the string s is empty, it is always a subsequence of any string t, so the function should return true. If t is empty but s is not, the function should return false. Additionally, if the length of s is greater than the length of t, the function should also return false. If all characters of s appear in t but are in the wrong order, the function should return false.

Complexity Analysis

The time complexity of the two-pointer approach is O(n), where n is the length of t, as we are iterating through t once. The binary search approach, on the other hand, has a time complexity of O(m + k log m), where m is the length of t, and k is the length of s. In terms of space complexity, the two-pointer method is O(1), as it only uses a constant amount of extra space. The binary search method requires O(m) space to store the indices of characters in t.

Alternative Approaches

While the two-pointer and binary search methods are the most efficient approaches for this problem, other methods such as dynamic programming or recursion could be considered. However, dynamic programming would be overkill for this problem since it requires building a table to track subsequences, which introduces unnecessary complexity for this task. Similarly, a backtracking recursive approach would be inefficient for large inputs.

Final Thoughts

The "Is Subsequence" problem highlights the importance of using efficient algorithms for string processing. The two-pointer approach is ideal for checking a single subsequence, while the binary search with a preprocessing approach excels in handling multiple queries. Understanding these techniques can be useful in solving related problems such as the Longest Common Subsequence or Edit Distance. For problems involving multiple queries, preprocessing and efficient lookups can significantly improve performance, making them key concepts in optimizing string processing tasks.