Find First Occurrence in String

Problem Statement

The LeetCode problem 'Find the Index of the First Occurrence in a String' requires the implementation of a function that will search for the first occurrence of a specific substring within a given string and return the index of the first character of that substring.

Function Description:

The function should take two parameters:

- text (string): The input text where we need to find the first occurrence of the substring.
- pattern (string): The substring that we are looking for within the input text.

Input:

- The input consists of two strings: text and pattern.
- The text string can be of varying length, containing alphanumeric characters as well as special symbols.
- The pattern string is the substring that we are searching for within the text string.

Output:

- The function should return an integer representing the index of the first character of the first occurrence of the pattern in the text string.
- If the pattern is not found in the text, the function should return -1.

Notes:

- If the pattern is an empty string, the function should return 0 as the index.
- The search for the pattern should be case-sensitive.
- If the pattern occurs multiple times in the text, the function should still return the index of the first occurrence.

Considering edge cases where the input strings are empty or contain special characters, the function should handle these scenarios gracefully and provide the correct output as per the defined requirements.

Examples

Here are several examples demonstrating the typical usage of the function to find the index of the first occurrence of a substring within a given text:

Example 1:

- Input:
  - text: "hello world"
  - pattern: "lo"
- Expected Output: 3
- Explanation: The first occurrence of "lo" in "hello world" starts at index 3.

Example 2:

- Input:
  - text: "programming is fun"
  - pattern: "gram"
- Expected Output: 3
- Explanation: The first occurrence of "gram" in "programming is fun" starts at index 3.

Example 3:

- Input:
  - text: "example"
  - pattern: "amp"
- Expected Output: -1
- Explanation: The pattern "amp" is not found in the text "example", so the function should return -1.

Example 4:

- Input:
  - text: "This is a test"
  - pattern: "test"
- Expected Output: 10
- Explanation: The first occurrence of "test" in "This is a test" starts at index 10.

These examples showcase various scenarios where the function should accurately identify the index of the first occurrence of a specified substring within a given text.

Constraints

When addressing the constraints associated with the LeetCode problem of finding the index of the first occurrence of a substring within a given string, it is essential to consider several limitations and special conditions that affect the implementation of the solution.

Input Size Limitations:

- The input strings, both text and pattern, can vary in length, potentially ranging from empty strings to strings containing a large number of characters.

- However, there may be constraints on the maximum length of these strings based on the platform or programming language requirements.
- Implementations should consider memory constraints when handling inputs of significant sizes to ensure efficient processing.

Special Conditions for Inputs:

- The pattern string can be empty, which requires a specific handling approach to return 0 as the index in such cases.
- The input strings may contain a combination of alphanumeric characters and special symbols, necessitating robust string processing techniques to accurately locate the pattern within the text.
- Case sensitivity in the search operation is a critical consideration, as the function should differentiate between uppercase and lowercase characters when matching the pattern within the text.

Assumptions:

- The problem statement assumes that the input strings provided to the function are valid and correctly formatted for the search operation.
- It is assumed that the function will always receive valid inputs conforming to the expected data types (strings) and will not encounter unexpected data formats during execution.
- The function is expected to return -1 if the specified pattern is not found within the text string, indicating the absence of the substring.

By outlining these constraints associated with input sizes, special conditions for inputs, and underlying assumptions made in the problem statement, developers can create a robust solution that effectively addresses the requirements of the LeetCode problem while ensuring the functionality is reliable across various scenarios.

Explanation of Approaches for Solving the Problem

When tackling the challenge of finding the index of the first occurrence of a substring within a given string, there are several strategies and algorithm choices to consider. Each approach has its strengths and weaknesses, making some methods more suitable than others based on the specific requirements and constraints of the problem.

Naive Approach:

One straightforward method to solve this problem is the naive approach, which involves iterating through the text string character by character and checking if the pattern matches at each position. This method has a time complexity of O(n*m), where n is the length of the text and m is the length of the pattern. While simple to implement, the naive approach may not be the most efficient solution, especially for large texts and patterns.

Knuth-Morris-Pratt Algorithm:

For more optimal performance, the Knuth-Morris-Pratt (KMP) algorithm offers an efficient solution to finding the first occurrence of a substring within a string. The KMP algorithm preprocesses the pattern to determine the positions to skip in the text, thereby reducing unnecessary comparisons. With a time complexity of O(n + m), where n is the length of the text and m is the length of the pattern, the KMP algorithm provides faster execution compared to the naive approach, particularly for longer texts and patterns.

Boyer-Moore Algorithm:

Another effective algorithm for this problem is the Boyer-Moore algorithm, which focuses on skipping comparisons by analyzing the mismatched characters and shifting the pattern efficiently. By leveraging heuristic rules to skip unnecessary checks, the Boyer-Moore algorithm offers excellent performance in practice. With an average-case time complexity of O(n/m) for typical text and pattern combinations, Boyer-Moore can outperform both the naive approach and KMP algorithm in certain scenarios.

Rabin-Karp Algorithm:

The Rabin-Karp algorithm introduces a hashing technique to compare the pattern with substrings of the text efficiently. By calculating hash values for the pattern and sliding window substrings of the text, the Rabin-Karp algorithm can quickly identify potential matches before performing actual character comparisons. This algorithm is particularly useful when handling multiple pattern searches within the same text. The time complexity of the Rabin-Karp algorithm is O(n + m) on average, making it a competitive choice for various text and pattern lengths.

Choosing the Right Approach:

When selecting an approach to find the index of the first occurrence of a substring in a string, it is essential to consider the characteristics of the input data, such as the length of the text and pattern, the frequency of pattern searches, and the need for optimal performance. While the naive approach provides a simple solution, it may not scale well for large inputs. In contrast, advanced algorithms like KMP, Boyer-Moore, and Rabin-Karp offer improved efficiency and are better suited for handling significant text processing tasks.

By evaluating the strengths and weaknesses of each algorithm and considering the specific requirements of the problem, developers can choose the most appropriate approach to efficiently solve the challenge of finding the index of the first occurrence of a substring within a given string.

Simplified Explanation

To find the index of the first occurrence of a word within a sentence, we need to look for that specific word and determine where it starts. Here's a simple breakdown of how this process works:

Reading the Sentence:

- Imagine we have a sentence like "hello world" and we want to find the word "world" in it.

Looking for the Word:

- We start by scanning each letter in the sentence to see if it matches the first letter of the word we are searching for.

Matching the Word:

- Once we find a matching letter, we check the following letters to see if they form the complete word we are looking for.

Identifying the Start:

- When we find a sequence of letters that match our word, we note the position where this sequence begins as the index.

Returning the Result:

- Finally, we return this index as the location where the word starts in the sentence.

If the word is not found in the sentence, we simply return -1 to indicate that it is not present. It's important to remember that we are looking for exact matches, including uppercase and lowercase letters. Additionally, if the word we are searching for is empty, we return 0 as the index.

By following these steps and understanding how to scan a sentence for a specific word, we can effectively determine the starting position of that word within the given text.

Python Code for Finding the Index of the First Occurrence in a String

```
class Solution:

    def strStr(self, haystack: str, needle: str) -> int:

        if not needle:

            return 0  # An empty needle is found at index 0


        h_len = len(haystack)  # Get the length of haystack

        n_len = len(needle)  # Get the length of needle


        for i in range(h_len - n_len + 1):  # Loop through the haystack to
find the needle

            if haystack[i:i + n_len] == needle:  # Check if the substring from
i to i + n_len is equal to needle

                return i  # Return the start index of the first occurrence


        return -1  # If needle is not found, return -1

# Example Usage
text = "hello world"
pattern = "lo"
print(find_first_occurrence(text, pattern))  # Output: 3
```

This Python function find_first_occurrence takes a text and a pattern as input and returns the index of the first occurrence of the pattern in the text. Here's a breakdown of the code:

1. The function checks if the pattern is empty and returns 0 if it is, as per the problem requirements.
2. It then iterates through the text to find the first occurrence of the pattern.
3. If the pattern is found, it returns the index where the pattern starts in the text.
4. If the pattern is not found, the function returns -1 to indicate that the pattern is not present in the text.

This well-documented Python code provides a clear and concise solution to the problem of finding the index of the first occurrence of a substring within a given string, addressing the specified requirements effectively.

Explanation of Each Line of Code

Let's break down the Python code provided for finding the index of the first occurrence of a substring in a given text. The function find_first_occurrence is designed to efficiently locate the starting index of a specified substring within a text string and return the appropriate result based on the search outcome.

Code Breakdown:

Function Definition:

- This line defines a function named find_first_occurrence that takes two parameters: text (the input text) and pattern (the substring to search for).

Docstring Description:

- The docstring provides a clear explanation of the function's purpose, input parameters, and expected return value to guide users on how to use the function effectively.

Handle Empty Pattern Case:

- This conditional check ensures that if the pattern provided is empty, the function immediately returns 0 as per the problem requirement, indicating that the pattern is found at index 0.

Pattern Search Loop:

- The function iterates through the text string up to a point where the remaining characters are equal to or longer than the pattern length.
- It checks if the substring of text starting at index i and ending at i + len(pattern) matches the pattern.
- If a match is found, the function returns the starting index i where the pattern is first encountered in the text.

Pattern Not Found:

- If the loop completes without finding the pattern, the function returns -1 to indicate that the pattern is not present in the given text.

Example Usage:

- This example demonstrates how to use the function by providing a sample text and pattern, and then printing the result of the function call, showcasing the expected output.

By comprehensively explaining each line and section of the Python code, users can grasp the functionality and logic behind the implementation of the find_first_occurrence function for efficiently solving the task of locating the first occurrence of a specific substring within a given text.

```
text = "hello world"
pattern = "lo"
print(find_first_occurrence(text, pattern))  # Output: 3

return -1

for i in range(len(text) - len(pattern) + 1):
    if text[i:i + len(pattern)] == pattern:
        return i

if not pattern:
    return 0

"""
Function to find the index of the first occurrence of a substring in a given
text.

Args:
text (str): The input text where we need to find the first occurrence of the
substring.
pattern (str): The substring that we are looking for within the input text.

Returns:
int: Index of the first character of the first occurrence of the pattern in
the text string. Returns -1 if pattern is not found.
"""

def find_first_occurrence(text, pattern):
```