Longest Common Prefix

When dealing with an array of strings, finding the longest common prefix can be a crucial task. The longest common prefix is the string that all strings in the array share as a prefix. In other words, it is the longest sequence of characters that is common among all strings at the beginning.

To find the longest common prefix among a set of strings, one common approach is to compare characters at the same index across all the strings. Starting from the first character, we check if it is the same for all strings. If it is, we move on to the next character; if not, we stop and return the prefix found so far.

This process continues until a mismatch is encountered or until we reach the end of any string. The characters checked along the way form the longest common prefix.

Efficient algorithms for finding the longest common prefix often involve sorting the array of strings. By sorting the strings, we can easily compare the first and last strings to determine the common prefix they share. This approach helps in reducing the overall time complexity of the operation.

Understanding how to efficiently find the longest common prefix can be valuable in various applications, such as searching, autocomplete functionalities, and data processing tasks. By implementing the appropriate algorithm, one can optimize the process of identifying common prefixes within a set of strings.

Problem Statement

The task at hand is to write a function that can efficiently determine the longest common prefix string among an array of strings. In this scenario, the common prefix refers to the sequence of characters that all strings in the array share at the beginning. If there is no common prefix among the strings, the function should return an empty string ("").

To achieve this, the function needs to compare characters at the same index across all the strings in the array. The comparison process starts from the first character of each string, checking if they are the same for all strings. If there is a match, the function proceeds to the next character; otherwise, it stops and returns the prefix found up to that point.

This comparison continues until a mismatch is detected or until the function reaches the end of any string in the array. The characters examined during this process collectively form the longest common prefix shared by all strings.

Implementing an efficient algorithm to find the longest common prefix often involves sorting the array of strings. Sorting simplifies the process as it allows for easy comparison between the first and last strings in the sorted array to

identify the common prefix they share. This sorting technique significantly contributes to reducing the overall time complexity of the operation.

The ability to identify the longest common prefix among a set of strings is crucial in various applications, including search functionalities, autocomplete features, and data processing tasks. By developing and integrating an optimized algorithm for this task, one can streamline the process of identifying common prefixes within a collection of strings.

Examples

Here are some examples illustrating different cases of input and output for the longest common prefix problem:

Input: strs = ["flower","flow","flight"]

- **Output: "fl"**

Input: strs = ["dog","racecar","car"]

- **Output: ""**
- **Explanation: There is no common prefix among the input strings.**

Input: strs = ["apple","ape","apricot"]

- **Output: "ap"**

Input: strs = ["coding","code","coder"]

- **Output: "code"**

Input: strs = ["book","boot","box"]

- **Output: "bo"**

These examples showcase various scenarios where the function for finding the longest common prefix operates on different arrays of strings. The outputs demonstrate the shared prefixes among the input strings, highlighting the effectiveness of the algorithm in identifying common prefixes efficiently.

Constraints

When developing the function to determine the longest common prefix among an array of strings, several constraints need to be considered to ensure the validity and efficiency of the solution. The constraints that the input values will respect are outlined below:

- **(1 ≤ strs.length ≤ 200): The array of strings will contain a minimum of 1 string and a maximum of 200 strings.**

- **(0 ≤ strs[i].length ≤ 200): Each string in the array can have a length ranging from 0 to 200 characters.**
- **strs[i] consists of only lowercase English letters: Each string in the array will only contain lowercase English letters, ensuring uniformity in character types for comparison during the common prefix determination process.**

By adhering to these constraints, the function can effectively handle a wide range of input scenarios while maintaining consistency in the evaluation of common prefixes among the strings. The defined constraints provide clear boundaries for the input values, guiding the function's execution and optimizing its performance in identifying the longest common prefix shared by the array of strings.

Explanation Like You're 16

Understanding the Problem

So, imagine you have a bunch of words, and you want to find the part at the beginning that they all have in common. It's like finding the starting letters that are the same in all the words.

Step-by-Step Solution

1. **First, check if there are any words at all. If you don't have any words, then there is no common starting part, so the answer is just nothing.**
2. **Next, you put all the words in order. This just helps make things easier.**
3. **Now, you look at the first word and the last word. You compare them letter by letter until you find where they are different.**
4. **Whatever letters they have in common at the beginning are your answer. That's the common starting part for all the words.**

Let's Try an Example

Let's say we have these words: "apple," "ape," and "apricot."

- **First, we sort them: "ape," "apple," "apricot."**
- **Then, we compare "ape" and "apricot." The common part is "ap."**
- **So, our answer is "ap."**

That's basically how you find the longest common starting part among a group of words. It's like a fun puzzle to see what they all share at the beginning!

Full Code

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
            return ""
        strs.sort()
        first, last = strs[0], strs[-1]
```

```
    for i in range(min(len(first), len(last))):
        if first[i] != last[i]:
            return first[:i]
    return first[:min(len(first), len(last))]
```

In this section, the complete code for the function that efficiently determines the longest common prefix string among an array of strings is provided. The code snippet above showcases the implementation of the longestCommonPrefix **function using Python.**

The function takes a list of strings strs **as input and returns the longest common prefix shared by all the strings. Here's a breakdown of how the code works:**

- **The function first checks if the input list of strings** strs **is empty. If the list is empty, it returns an empty string as there is no common prefix to find.**
- **The function then sorts the list of strings in lexicographical order using the** sort **method.**
- **It then initializes two variables** first **and** last **to store the first and last strings after sorting.**
- **Next, the function iterates through the characters of the first and last strings simultaneously using a for loop and compares them character by character.**
- **If a mismatch is found at any index, the function returns the prefix found so far using slicing operations.**
- **If no mismatch is encountered and the loop completes, the function returns the full common prefix found up to the length of the shortest string between the first and last strings.**

By following this algorithm, the function efficiently identifies the longest common prefix shared by all strings in the input array, providing a concise and optimized solution to the longest common prefix problem in Python.

Code Explanation

In this section, we will delve into a detailed explanation of the code provided for efficiently determining the longest common prefix string among an array of strings. The code snippet below showcases the implementation of the longestCommonPrefix **function using Python.**

Check for Empty List:

This initial check ensures that if the list of strings is empty, the function immediately returns an empty string since there is no common prefix to determine.

Sort the List:

Sorting the list of strings in lexicographical order facilitates the comparison process and simplifies identifying the common prefix shared by all strings.

Take the First and Last Strings:

After sorting the list, the first and last strings represent the strings that will help determine the common prefix as they encompass the minimum and maximum lexicographical order.

Compare Characters:

By iterating through the characters of the first and last strings simultaneously, the function compares characters at each index. If a difference is found, the function returns the common prefix up to that point.

Return Full Length if All Match:

If all characters match without encountering a difference between the first and last strings, the function returns the full common prefix up to the length of the shortest string.

This structured breakdown of the code highlights the step-by-step process involved in efficiently identifying the longest common prefix among a set of strings. Each segment of the code serves a specific purpose in the overall algorithm, contributing to a streamlined and optimized solution for the longest common prefix problem in Python.

Final Thoughts

In conclusion, the approach described for efficiently determining the longest common prefix among an array of strings proves to be a robust and effective method. By leveraging sorting techniques and comparing the first and last strings in the sorted array, this approach streamlines the process of identifying common prefixes and handles various edge cases seamlessly.

The key benefits of this approach lie in its ability to reduce the overall time complexity of the operation by simplifying the comparison process. Sorting the strings allows for a systematic evaluation of the common prefix, ensuring a more efficient and optimized solution.

Furthermore, this method excels in handling edge cases such as empty lists and arrays with varied string lengths. The initial check for an empty list ensures a prompt return of an empty string when no common prefix exists, while the comparison between the first and last strings accommodates arrays with differing string lengths, providing a comprehensive solution for a wide range of scenarios.

By implementing this approach, developers can enhance the performance of common prefix identification tasks in applications requiring string manipulation and data processing. The optimized algorithm not only enhances the efficiency of finding common prefixes but also contributes to the overall effectiveness of search functionalities, autocomplete features, and other string-related operations.

In mastering this approach, developers can unlock a valuable tool for streamlining string processing tasks and improving the accuracy of common prefix determinations. Understanding the nuances of this method and its benefits can empower individuals to tackle string-related challenges with confidence and precision, ultimately enhancing the functionality and performance of their applications.

```
return first[:min(len(first), len(last))]

for i in range(min(len(first), len(last))):
    if first[i] != last[i]:
        return first[:i]

first, last = strs[0], strs[-1]

strs.sort()

if not strs:
    return ""
```