14. Longest Common Prefix

Easy Top String

Problem Description

The goal of this problem is to write a function that takes an array of strings and returns the longest common prefix that is shared among all the strings. The common prefix is the starting part of the strings that is identical for all of them. For example, if the input is `["flower","flow","flight"]`, the longest common prefix is `"fl"`, since it's the beginning part that all the strings have in common. If the strings have no common prefix, the function should return an empty string `""`.

The function will need to check the strings in the array and compare them to find the longest sequence of characters that is present at the start of each string. This problem requires an efficient way to compare the strings and determine the common prefix.

Intuition

When trying to find a common pattern among a set of items, a logical step is to begin by looking at the first item and then comparing it with subsequent items. Similarly, in the case of finding the longest common prefix, it makes sense to start with the first string in the array and compare its characters one by one with the characters at the corresponding positions in the other strings.

To achieve this, we perform the following steps:

1. If the array is empty, there can't be a common prefix, so we would return an empty string. However, if the array contains at least one string, then we consider the entire first string as the initial longest common prefix candidate.
2. Next, we iterate through the characters of the first string. For each character, we check if the same character index in every other string in the array is the same character. To do this, we have an inner loop that compares the character at index `i` of `strs[0]` with the character at index `i` of each subsequent string (`strs[1:]`).
3. If at any point the comparison fails because we've reached the end of one of the other strings, or the characters do not match, we've found the end of the common prefix. We can immediately return the substring of the first string up to (but not including) character `i`.
4. If the inner loop completes without finding a mismatch, it means that the current character is part of the common prefix for all strings checked so far. We continue moving to the next character.
5. If we successfully compare all characters in the first string without finding a mismatch, then the entire first string is the common prefix. In this case, the simple conclusion is that there is no shorter prefix than the first string itself, so we return it.

By following this comparison logic, we ensure that as soon as a discrepancy is found, we do not waste any further time checking additional characters, thereby optimizing the function for better performance.

Solution Approach

The solution employs a simple yet effective algorithm that involves iterating over each character position of the strings in the array and comparing them to find the longest common prefix.

Here's how the implementation works step by step:

1. We start by looking at the first string in the array, `strs[0]`, and assume that it could be the potential longest common prefix. We use this string as a reference to compare with all other strings.
2. We then enter a for-loop, which iterates over the character indices of the first string. The variable `i` represents the index of the character we are currently checking.
3. Within this loop, we initiate another for-loop that iterates through the remaining strings in the array—`strs[1:]`.
4. For each string `s` in this inner loop, we perform two checks: a. If the current index `i` is greater than or equal to the length of the string `s`, this means we have reached the end of this string, and thus, it cannot have a longer common prefix. b. If the character at index `i` of the string `s` does not match the character at the same index in `strs[0]`, this indicates that the current character does not form part of a common prefix.
5. If either condition in the inner loop is true, we have identified the end of the common prefix, and we return the substring of the first string from index `0` to `i`, using `s[:i]`. This is the longest common prefix found up to that point.
6. If the inner loop completes without triggering the return statement, it means that the current character at index `i` is shared by all strings checked, and thus, the loop continues to the next character.
7. If the outer loop completes without any return, this indicates that every character of the first string `strs[0]` is part of the common prefix for all strings. Therefore, we can safely return `strs[0]` as the longest common prefix.

This algorithm essentially implements a character-by-character comparison, making use of string slicing for efficient checking and early stopping as soon as a mismatch is found. No additional data structures are needed, thus the space complexity is kept to a minimum.

The early return mechanism greatly improves performance by terminating the comparison as soon as the longest common prefix is established, without further unnecessary comparisons.

The procedure can be visualized as lining up all the strings vertically and scanning down the columns (indices). Once a discrepancy is found in any column, we know that the common prefix can only be as long as the last column without discrepancies.

Example Walkthrough

To illustrate the solution approach, let's take an example array of strings: `["cardboard", "cart", "carrot", "carbon"]`. We want to find the longest common prefix among these strings.

1. **Initial Assumption**: We start with the assumption that the first string "cardboard" is our longest common prefix candidate.
2. **Character-by-Character Comparison**:
   - We start comparing the first character 'c' from "cardboard" with the first character of all the other strings. All strings have 'c' as their first character.
   - Next, we compare the second character 'a' from "cardboard" with the second character of the other strings. All strings have 'a' as their second character.
   - We continue to the third character 'r' from "cardboard" and compare it with the third character of the other strings. All strings have 'r' as well.
3. **Mismatch and Early Termination**:
   - Now we move to the fourth character 'd' from "cardboard". However, when we compare it with the fourth character of "cart", we find that it is 't' and not 'd'. This is a mismatch.
   - Since "cart" has no more characters to compare (it is shorter than "cardboard"), this is our signal to stop.
4. **Returning the Result**: At this point, we found that all strings share the prefix "car". We return this prefix as it is the longest common prefix that can be formed from all given strings.

The longest common prefix based on our algorithm is "car". This was determined efficiently by stopping comparisons once the first mismatch occurred or a string ended, ensuring no extra work was done.

Solution Implementation

Python

```python
class Solution:
    def longestCommonPrefix(self, strings: List[str]) -> str:
        # Assumes the input list of strings is not empty
        # The outer loop goes through each character of the first string
        for index in range(len(strings[0])):
            # Inner loop checks the character at the current position across all other strings
            for string in strings[1:]:
                # Checks if we've reached the end of any string or a character mismatch is found
                if index >= len(string) or string[index] != strings[0][index]:
                    # Return the longest common prefix found so far
                    return strings[0][:index]
        # If no early return happened, the first string itself is the common prefix
        return strings[0]
```

Java

```java
class Solution {

    // Method to find the longest common prefix from an array of strings.
    public String longestCommonPrefix(String[] strs) {
        int numberOfStrings = strs.length; // Total number of strings in the array.

        // Loop through each character of the first string.
        for (int index = 0; index < strs[0].length(); ++index) {
            // Compare the character with the same position in the remaining strings.
            for (int stringIndex = 1; stringIndex < numberOfStrings; ++stringIndex) {
                // Check two conditions here:
                // 1. If the current string is shorter than the current character index, or
                // 2. If the current character does not match the character in the first string.
                // In either case, that means we've found the end of the common prefix.
                if (strs[stringIndex].length() <= index || strs[stringIndex].charAt(index) != strs[0].charAt(index)) {
                    // Therefore, we return the substring from the start to the current index from the first string.
                    return strs[0].substring(0, index);
                }
            }
        }

        // If we manage to check all characters of the first string without finding a mismatch,
        // it means that the entire first string is a common prefix.
        return strs[0];
    }
}
```

C++

```cpp
class Solution {
public:
    // This function finds the longest common prefix among the strings in the given vector.
    string longestCommonPrefix(vector<string>& strs) {
        int numberOfStrings = strs.size(); // Total number of strings in the vector.

        // Loop over the characters of the first string.
        for (int charIndex = 0; charIndex < strs[0].size(); ++charIndex) {
            // Compare the current character with the same position in each subsequent string.
            for (int strIndex = 1; strIndex < numberOfStrings; ++strIndex) {
                // Check if the current character index exceeds the length of the current string
                // or the characters do not match.
                if (strs[strIndex].size() <= charIndex || strs[strIndex][charIndex] != strs[0][charIndex]) {
                    // Return the longest common prefix found so far.
                    return strs[0].substr(0, charIndex);
                }
            }
        }

        // If we reach this point, it means the first string is a common prefix for all strings in the array.
        // Thus, simply return the first string.
        return strs[0];
    }
};
```

TypeScript

```typescript
function longestCommonPrefix(strs: string[]): string {
    // Find the shortest string length, as the common prefix can't be longer than that
    const shortestLength = strs.reduce((minLength, currentString) => Math.min(minLength, currentString.length), Infinity);

    // Starting from the length of the shortest string, reduce the length
    // until we find the longest common prefix
    for (let i = shortestLength; i > 0; i--) {
        // Grab the prefix from the start of the first string up to the current length
        const currentPrefix = strs[0].substring(0, i);

        // Check if every string in the array has the same prefix
        if (strs.every(string => string.startsWith(currentPrefix))) {
            // If they do, return the current prefix
            return currentPrefix;
        }
    }

    // If there's no common prefix at all, return an empty string
    return '';
}


class Solution:
    def longestCommonPrefix(self, strings: List[str]) -> str:
        # Assumes the input list of strings is not empty
        # The outer loop goes through each character of the first string
        for index in range(len(strings[0])):
            # Inner loop checks the character at the current position across all other strings
            for string in strings[1:]:
                # Checks if we've reached the end of any string or a character mismatch is found
                if index >= len(string) or string[index] != strings[0][index]:
                    # Return the longest common prefix found so far
                    return strings[0][:index]
        # If no early return happened, the first string itself is the common prefix
        return strings[0]
```

Time and Space Complexity

The time complexity of the code is `O(n * m)`, where `n` is the number of strings in the given list, and `m` represents the length of the shortest string in the list. Each character of the shortest string is iterated `n` times to check if it is a common prefix among all the strings.

The space complexity of the solution is `O(1)`, as it does not allocate additional space proportional to the input size, using only a few variables to perform the checks and return the result.