longest common prefix leetcode solution

longest common prefix leetcode solution is a common problem frequently encountered in coding interviews and algorithm challenges. It involves finding the longest prefix string that is common to all strings in a given array. Understanding and implementing efficient solutions to this problem is essential for optimizing string manipulation tasks and improving problem-solving skills. This article explores various approaches to the longest common prefix problem, including straightforward brute-force methods and more sophisticated techniques such as divide and conquer, binary search, and vertical scanning. Each method will be explained with detailed insights into time and space complexities, helping programmers select the best approach for their needs. The discussion also covers practical tips on coding the longest common prefix LeetCode solution in programming languages like Python and Java. Following this introduction, the article presents a comprehensive table of contents for easy navigation through the different sections addressing the longest common prefix LeetCode solution.

Understanding the Longest Common Prefix Problem

The longest common prefix problem requires identifying the longest substring at the beginning of each string in an array that remains consistent across all elements. In other words, the goal is to find the largest prefix shared by every string in the list. This problem is commonly presented in coding platforms such as LeetCode under the title "Longest Common Prefix," making it a popular challenge for developers to demonstrate proficiency in string manipulation and algorithm design.

Handling edge cases, such as empty strings or arrays with a single element, is critical in providing a robust longest common prefix LeetCode solution. Additionally, understanding the problem constraints, such as the maximum length of strings and the number of strings in the array, influences the choice of algorithm for an efficient and scalable solution.

Brute Force Approach

The brute force method for solving the longest common prefix problem is the most straightforward and intuitive. It involves comparing characters of the first string with each subsequent string until a mismatch is found. This approach progressively reduces the prefix length until the common prefix is identified or eliminated.

How the Brute Force Approach Works

Starting with the entire first string as a candidate prefix, the algorithm iterates through each character position and compares it against the corresponding character in all other strings. If a mismatch occurs or the end of any string is reached, the prefix is truncated to the current matched length. This process continues until the longest common prefix is determined.

- Initialize the prefix as the first string.
- Iterate through the rest of the strings in the array.
- Compare characters at each position with the prefix.
- Reduce the prefix length when a mismatch occurs.
- Return the final prefix as the longest common prefix.

Although simple, the brute force approach can be inefficient for large datasets, with a worst-case time complexity of O(S), where S is the sum of all characters in all strings.

Vertical Scanning Method

The vertical scanning technique improves the brute force method by comparing characters column by column rather than string by string. This approach inspects each character index across all strings simultaneously, stopping when a mismatch or the end of any string is encountered.

Advantages of Vertical Scanning

Vertical scanning offers a cleaner and often faster way to find the longest

common prefix, especially when prefixes are short or strings have significant variance early on. It minimizes unnecessary comparisons by focusing on character positions rather than full string comparisons.

- Scan each character index vertically across all strings.
- Stop immediately if a mismatch is found or the end of a string is reached.
- Return the substring from the first string up to the matched index.

This method maintains a time complexity of O(S) but often performs better due to early termination in practical scenarios.

Divide and Conquer Strategy

The divide and conquer approach splits the array of strings into two halves, recursively finds the longest common prefix in each half, and then merges the results. This method leverages recursion to break down the problem into smaller, manageable subproblems.

Implementing Divide and Conquer

The algorithm recursively divides the input until it reaches a base case of a single string or an empty array. It then combines the results by finding the common prefix between two halves. This combination step is performed by comparing characters until a mismatch is found.

- Divide the array into two halves.
- Recursively find the longest common prefix of each half.
- Merge the two prefixes by comparing characters.
- Return the merged prefix as the result for the current recursion level.

Divide and conquer typically achieves a time complexity of O(S), but it offers better structure and clarity in code, especially for larger datasets.

Binary Search Technique

The binary search approach applies a search on the length of the prefix rather than the strings themselves. It uses the minimum string length in the array as the upper bound and performs binary search to check if a prefix of a certain length is common to all strings.

Steps in the Binary Search Method

This method involves checking the validity of a prefix length by comparing substrings from all strings. Based on the check, the search space is adjusted until the maximum valid prefix length is found.

- Find the minimum string length in the array.
- Set low and high pointers for binary search on prefix length.
- Check if all strings share the prefix of the middle length.
- Adjust low or high based on the validity of the prefix.
- Return the prefix corresponding to the maximum valid length.

The binary search approach typically runs in O(S * log m), where m is the minimum string length, offering a balanced trade-off between time and complexity.

Trie Data Structure Approach

Using a Trie (prefix tree) is an advanced method for solving the longest common prefix problem efficiently, especially when handling many strings with overlapping prefixes. This data structure stores characters in a tree format, allowing quick access to common prefixes.

Building and Using a Trie for the Solution

The algorithm inserts all strings into the Trie, then traverses from the root node down the tree until a node with more than one child or a terminal node is encountered. The characters along this path form the longest common prefix.

- Create a Trie node class with children and end-of-word flag.
- Insert all strings into the Trie character by character.
- Traverse the Trie from the root to find the longest path with a single child.
- Record the characters along this path as the longest common prefix.

While a Trie provides efficient prefix queries, it uses additional memory, and the overall time complexity is O(S), where S is the total number of characters.

Time and Space Complexity Analysis

Evaluating the time and space complexities of different longest common prefix LeetCode solutions helps in selecting the optimal algorithm based on problem constraints. Each approach offers a trade-off between simplicity, speed, and memory usage.

Comparative Complexity Overview

Most common approaches have a time complexity of O(S), where S is the sum of all characters in the input array. However, binary search introduces a logarithmic factor, and Trie-based solutions require extra space for tree nodes.

- **Brute Force:** Time - O(S), Space - O(1)
- **Vertical Scanning:** Time - O(S), Space - O(1)
- **Divide and Conquer:** Time - O(S), Space - O(m * log n) due to recursion stack
- **Binary Search:** Time - O(S * log m), Space - O(1)
- **Trie:** Time - O(S), Space - O(S)

Understanding these complexities allows developers to choose the most efficient solution based on input size and available resources.

Practical Implementation Tips

Implementing the longest common prefix LeetCode solution efficiently requires attention to detail, handling edge cases, and writing clean, readable code. Optimizing string comparisons and avoiding unnecessary operations can significantly improve performance.

Best Practices for Coding the Solution

- Check for empty arrays or single-element arrays before processing.
- Use built-in string methods where possible for substring extraction.
- Terminate loops early upon detecting mismatches to save processing time.
- Consider input size when choosing the algorithm to balance speed and memory usage.
- Write modular code with helper functions for clarity and

maintainability.

Thorough testing with various inputs, including edge cases like arrays with empty strings or identical strings, ensures robustness and correctness of the longest common prefix solution.

Frequently Asked Questions

What is the problem statement of the Longest Common Prefix on LeetCode?

The Longest Common Prefix problem on LeetCode asks to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string.

What is a simple approach to solve the Longest Common Prefix problem?

A simple approach is to compare characters of the strings one by one from the start until a mismatch is found or the shortest string ends. The common characters up to that point form the longest common prefix.

How can sorting the array help in solving the Longest Common Prefix problem?

By sorting the array, the common prefix of the entire array must be a prefix of the first and last strings. So, comparing only the first and last strings after sorting can give the longest common prefix efficiently.

What is the time complexity of the vertical scanning method for Longest Common Prefix?

The vertical scanning method compares characters column-wise and has a time complexity of O(S), where S is the sum of all characters in all strings.

How can divide and conquer be applied to the Longest Common Prefix problem?

Divide and conquer splits the array into two halves, recursively finds the longest common prefix for each half, and then finds the common prefix between these two results.

What data structures can be used to optimize the Longest Common Prefix solution?

A Trie (prefix tree) can be used to efficiently store prefixes of all strings, allowing quick retrieval of the longest common prefix.

Can binary search be used to find the Longest Common Prefix? How?

Yes, binary search can determine the longest common prefix length by checking the middle length prefix for all strings and adjusting the search range until the maximum prefix length is found.

What is the space complexity of the Trie-based approach for the Longest Common Prefix problem?

The space complexity is O(S), where S is the sum of all characters in the input strings, because the Trie stores all characters of the input strings.

Why might the horizontal scanning method be less efficient for large inputs?

Horizontal scanning compares prefixes between strings sequentially and may result in repeated comparisons, leading to higher time complexity, especially when many strings share small prefixes.

Is it necessary to handle edge cases like empty string arrays or single string arrays in the Longest Common Prefix problem?

Yes, handling edge cases is important. For an empty array, the result should be an empty string. For a single string array, the longest common prefix is the string itself.

Additional Resources

1. *Mastering String Algorithms: LeetCode Solutions for Longest Common Prefix*
This book dives deep into string manipulation techniques with a focus on the Longest Common Prefix problem commonly seen on LeetCode. It provides step-by-step solutions, optimized algorithms, and practical coding examples. Readers will learn both brute force and advanced approaches, enhancing their problem-solving skills in competitive programming.

2. *Efficient Coding Patterns: Solving Longest Common Prefix and Beyond*
Explore efficient coding patterns that simplify solving string-related challenges like the Longest Common Prefix. This book covers fundamental

concepts, common pitfalls, and optimization strategies. It is ideal for programmers looking to improve their algorithmic thinking and implement clean, performant code.

3. *Data Structures & Algorithms in Python: Longest Common Prefix Explained*
Designed for Python enthusiasts, this guide breaks down the Longest Common Prefix problem using intuitive explanations and code samples. It integrates foundational data structures such as tries and arrays to build a comprehensive understanding. The book also compares multiple solution methods, helping readers choose the best approach for their needs.

4. *LeetCode Patterns: String Matching and Longest Common Prefix Techniques*
This book categorizes common LeetCode string problems, highlighting the Longest Common Prefix as a key pattern. It provides pattern recognition tips, reusable code snippets, and complexity analysis. Aimed at developers preparing for coding interviews, it boosts confidence through practice problems and detailed walkthroughs.

5. *Algorithmic Thinking: From Brute Force to Optimized Longest Common Prefix Solutions*
Learn to transform naive solutions into highly efficient algorithms with this insightful resource. It walks readers through the evolution of solving the Longest Common Prefix problem, emphasizing time and space complexity improvements. The book encourages a mindset of continuous optimization applicable across various algorithmic challenges.

6. *Hands-On LeetCode: String Algorithms and Longest Common Prefix*
Packed with practical exercises, this hands-on guide focuses on implementing and testing Longest Common Prefix solutions. It stresses writing bug-free code and understanding edge cases. Readers gain confidence by building from basic to advanced problems, supplemented with tips for debugging and performance tuning.

7. *Cracking Coding Interviews: Longest Common Prefix and String Challenges*
Tailored for job seekers, this book demystifies string problems like Longest Common Prefix commonly asked in technical interviews. It includes proven strategies, time management advice, and detailed solution breakdowns. The content prepares candidates to tackle similar problems efficiently under pressure.

8. *Advanced String Algorithms: Tries, Prefix Trees, and Longest Common Prefix*
Delve into sophisticated data structures such as tries and prefix trees that are instrumental in solving Longest Common Prefix problems. This book explains their construction, traversal, and application with code examples. It is perfect for readers aiming to master advanced algorithmic concepts in string processing.

9. *Programming Challenges: Longest Common Prefix and Related String Problems*
This compilation of programming challenges centers on the Longest Common Prefix and other related string manipulation tasks. It encourages readers to apply different problem-solving techniques and compare their effectiveness.

With a mix of theoretical insights and practical exercises, the book serves as a valuable training tool for competitive programmers.

Longest Common Prefix Leetcode Solution

Longest Common Prefix Leetcode Solution

Related Articles

- main idea worksheet grade 2
- magic tree house fact tracker 13 pilgrims mary pope osborne
- lux programmable thermostat manual

Back to Home