An Efficient Algorithm for Finding the Longest Common Prefix

Ollie

October 12, 2025

Abstract

This study presents an efficient algorithm for solving the Longest Common Prefix problem. The algorithm leverages lexicographic ordering to reduce the problem to a comparison between only two strings. In here a formal correctness proof using loop invariants is provided, and the algorithm's time and space complexity is analyzed.

1 Introduction

The Longest Common Prefix problem is a fundamental string processing problem that asks: given an array of strings, find the longest string that is a prefix of all strings in the array. This problem has applications in text processing, autocompletion systems, and data compression.

The problem is formally defined on LeetCode[1] and admits several solutions. We present an elegant, sorting based approach that significantly reduces the comparison complexity.

2 Problem Formulation

Definition 1 (Longest Common Prefix)**.** Given an array of strings $S = \{s_1, s_2, \ldots, s_n\}$, the longest common prefix is the longest string p such that for all $i \in \{1, 2, \ldots, n\}$, p is a prefix of s_i.

3 Key Insight

The algorithm relies on a crucial observation about lexicographic ordering:
Lemma 1. *If an array of strings is sorted lexicographically, then the longest common prefix of all strings is equal to the longest common prefix of the first and last strings in the sorted array.*

Proof. Let $S' = \{s'_1, s'_2, \ldots, s'_n\}$ be the sorted array where s'_1 is lexicographically smallest and s'_n is lexicographically largest.

[1] `https://leetcode.com/problems/longest-common-prefix/description`

Suppose p is a common prefix of s'_1 and s'_n. For any intermediate string s'_i where $1 < i < n$, we have $s'_1 \leq s'_i \leq s'_n$ lexicographically. Since p is a prefix of both bounds, it must also be a prefix of s'_i.

Conversely, any prefix common to all strings must be a prefix of both s'_1 and s'_n. Therefore, the longest common prefix of all strings equals the longest common prefix of s'_1 and s'_n. □

4 Algorithm Description

The algorithm proceeds in three main steps:

1. Sort the array of strings lexicographically
2. Select the first and last strings from the sorted array
3. Compare characters position-by-position until a mismatch is found

4.1 Pseudocode

Algorithm 1 Longest Common Prefix

```
1:  procedure LongestCommonPrefix(strs)
2:      if strs is empty then
3:          return ""
4:      end if
5:      Sort(strs)                                  ▷ Lexicographic sort
6:      first ← strs[0]
7:      last ← strs[|strs| − 1]
8:      prefix ← ""
9:      m ← min(|first|, |last|)
10:     for i ← 0 to m − 1 do
11:         if first[i] = last[i] then
12:             prefix ← prefix + first[i]
13:         else
14:             break
15:         end if
16:     end for
17:     return prefix
18: end procedure
```

5 Correctness Proof

We establish correctness using a loop invariant for the character comparison loop.

5.1 Loop Invariant

Invariant Statement: At the start of each iteration i of the comparison loop, the first i characters of $first$ and $last$ are equal, and thus represent a prefix common to all strings in the array.

Formally: At the beginning of iteration i, the substring $first[0..i-1]$ equals $last[0..i-1]$, and this substring is a common prefix of all strings in the sorted array.

5.2 Proof of Correctness

Initialization: Before the first iteration ($i = 0$), the prefix considered is the empty string "". The empty string is trivially a prefix of all strings. Thus, the invariant holds initially.

Maintenance: Suppose the invariant holds at the start of iteration i. If $first[i] = last[i]$, then since the array is sorted lexicographically, all intermediate strings between $first$ and $last$ also share the same character at position i. Therefore, after extending the prefix by $first[i]$, the invariant continues to hold for iteration $i + 1$.

Termination: The loop terminates under two conditions:

1. A mismatch is found: $first[i] \neq last[i]$. Since $first$ and $last$ are the lexicographic extremes, no string can have a longer common prefix.
2. The end of the shorter string is reached: $i = \min(|first|, |last|)$. In this case, the shorter string itself is the longest common prefix.

In both cases, upon termination, $prefix$ contains exactly the longest common prefix of all strings in the array.

6 Complexity Analysis

6.1 Time Complexity

The algorithm's time complexity consists of two main components:

- **Sorting:** $O(n \log n)$ where n is the number of strings. More precisely, sorting strings has complexity $O(n \cdot m \log n)$ where m is the average string length, but we use the simplified bound.
- **Character comparison:** $O(m)$ where m is the length of the shortest string.

Overall Time Complexity: $O(n \log n + m)$

For most practical cases where $m \ll n \log n$, the sorting dominates, giving an effective complexity of $O(n \log n)$.

6.2 Space Complexity

The algorithm uses only a constant number of auxiliary variables ($first$, $last$, $prefix$, i, m) beyond the input array.

Space Complexity: $O(1)$ auxiliary space.
Remark 1. If the sorting algorithm requires additional space (e.g., merge sort), the space complexity would be $O(n)$. However, with in-place sorting algorithms like heapsort, we maintain $O(1)$ auxiliary space.

7 Implementation Considerations

The algorithm can be implemented in any programming language with built-in sorting capabilities. The key considerations are:

- Edge cases: empty array, array with single string, strings of different lengths
- Efficient string concatenation for building the prefix
- Choice of sorting algorithm based on the expected input characteristics

8 Conclusion

This study presented an efficient algorithm for finding the longest common prefix of an array of strings. By exploiting lexicographic ordering, we reduced the problem from comparing all strings to comparing only two extremal strings. The formal correctness proof via loop invariants ensures the algorithm's reliability, while the complexity analysis demonstrates its efficiency for practical applications.