3043. Find the Length of the Longest Common Prefix

Medium Tip Array Hash Table String

Problem Description

In this problem, we are given two arrays of positive integers, `arr1` and `arr2`. We are tasked with finding the length of the longest common prefix between any two integers, where one integer comes from `arr1` and the other comes from `arr2`.

A "prefix" in this context means the digits at the start of an integer, up until any length of that integer. For instance, 12 is a prefix of 123. If a prefix is found in both integers from `arr1` and `arr2`, it is considered a common prefix. Note that if the prefix doesn't exist in both integers, it's not common.

Our goal is to determine the size of the utmost common prefix discovered among all possible pairs made by taking one integer from each of the two arrays. If no common prefixes are found, then the answer we return should be 0.

Intuition

To solve this problem, we apply the concept of storing the prefixes in a hash table. The solution revolves around two main steps:

Firstly, we enumerate all possible prefixes from the integers in `arr1` and store each one in a set (which acts as our hash table).

Secondly, for every integer in `arr2`, we start with the integer itself and keep dividing it by 10 (which effectively removes the least significant digit) to generate its prefixes. Each generated prefix is checked against our hash table to see if it's a common prefix.

The logic behind gradually reducing the integer starting from its original value down to its high-order digits is that we are interested in the longest common prefix – so we start with checking the entire number and then reduce it from the right.

While we are checking for prefixes, we remember the length of the longest common prefix we've encountered by using a variable. This variable is updated anytime we find a longer common prefix than the current record.

This approach allows us to efficiently find the longest common prefix among all pairs by leveraging the quick lookup capabilities of a hash table.

Solution Approach

The solution can be broken down into two main phases, using the Python programming language:

1. Prefix Generation for `arr1`:
   - We initiate by creating an empty set `s`, which will be used to store the prefixes of the integers present in `arr1`.
   - For each number `x` in `arr1`, we start a loop where we keep adding `x` to our set `s`. Right after adding, we divide `x` by 10 using an integer division (`x //= 10`). This effectively removes the least significant digit of `x`, leaving us with a new prefix. We repeat this until `x` becomes 0, which means we've added all possible prefixes of `x` to the set.
2. Finding Longest Common Prefix with `arr2`:
   - We initialize an answer variable `ans` to 0. This will hold the length of the longest common prefix we find.
   - We iterate through each number `x` of `arr2`. Similar to the first phase, we keep dividing `x` by 10 to get its prefixes.
   - For each generated prefix of `x`, we check if this prefix exists in the set `s`. If it does, it means we have found a common prefix. We then compare the length of this prefix (by converting it to a string and getting the length) to our variable `ans`. If the length is greater, we update `ans` to the new length.
   - As soon as we find a matching prefix, we break out of the loop for the current number `x` because we're looking for the longest common prefix, and any further divisions will only produce shorter prefixes which are not of interest.

In summary, the solution utilizes a hash set for quick existence checks of prefixes, a loop to generate all possible prefixes of integers in `arr1`, and another loop to check prefixes of integers in `arr2` against this set. We also use the max function to update the length of the longest common prefix as we iterate through the elements of `arr2`. This approach guarantees that we find the longest common prefix in an efficient manner.

Example Walkthrough

Let's illustrate the solution with a small example using the following arrays:

- `arr1 = [12345, 678]`
- `arr2 = [123, 458]`

Prefix Generation for `arr1`

- Initialize set `s = {}`.
- For `12345` in `arr1`:
  - Add `12345` to `s`, now `s = {12345}`.
  - Iterate by dividing by 10 and adding to `s`: `1234`, `123`, `12`, `1`, until `x` becomes 0. `s` becomes `{12345, 1234, 123, 12, 1}`.
- For `678` in `arr1`:
  - Add `678` to `s`, now `s = {12345, 1234, 123, 12, 1, 678}`.
  - Continue dividing by 10 and adding to `s`: `67`, `6`.
  - Final `s = {12345, 1234, 123, 12, 1, 678, 67, 6}`.

Finding Longest Common Prefix with `arr2`

- Initialize `ans = 0`.
- For `123` in `arr2`:
  - `123` is in set `s`. A common prefix of length `3` is found.
  - Update `ans = 3` because it is greater than the current `ans` value.
  - No need to keep dividing `123` by 10 because we found the longest prefix with it.
- For `458` in `arr2`:
  - `458` is not in `s`.
  - Divide by 10, get `45`. `45` not in `s`.
  - Divide by 10, get `4`. `4` not in `s`.
  - All prefixes checked, move to next integer.

After iterating through both arrays, the longest common prefix we found was of length `3`. Hence, our answer is `3`.

```
Given `arr1 = [12345, 678]` and `arr2 = [123, 458]`, let's walk through the solution step by step.

1. Prefix Generation for `arr1`:
   - Create an empty set `s = {}`.
   - Add all prefixes of `12345` to `s`. We add `12345`, `1234`, `123`, `12`, and `1`.
   - Add all prefixes of `678` to `s`. We add `678`, `67`, and `6`.
   - Now, `s = {12345, 1234, 123, 12, 1, 678, 67, 6}`.

2. Finding Longest Common Prefix with `arr2`:
   - Initialize `ans = 0`.
   - Look at `123` from `arr2`. We find `123` in `s`. Update `ans` to `3` since `123` has length `3`.
   - Look at `458` from `arr2`. We do not find `458`, `45`, or `4` in `s`. We do not update `ans`.

The longest common prefix found is of length `3`, making our final answer `3`.
```

Solution Implementation

Python

```python
class Solution:
    def longestCommonPrefix(self, nums1: List[int], nums2: List[int]) -> int:
        # Initialize an empty set to store unique prefixes
        prefixes = set()

        # Loop through each number in the first list (nums1)
        for num in nums1:
            # Generate prefixes by continuously dividing the number by 10
            while num:
                # Add the current prefix to the set
                prefixes.add(num)
                # Update `num` by removing the last digit
                num //= 10

        # Initialize `max_prefix_length` to keep track of the longest prefix length
        max_prefix_length = 0

        # Loop through each number in the second list (nums2)
        for num in nums2:
            # Again, generate prefixes by continuously dividing the number by 10
            while num:
                # If the current prefix is in `prefixes` set, it is a common prefix
                if num in prefixes:
                    # Update `max_prefix_length` if this prefix is longer than the current max
                    max_prefix_length = max(max_prefix_length, len(str(num)))
                    # Break the loop as we've found the longest prefix for this number
                    break
                # Update `num` by removing the last digit
                num //= 10

        # Return the length of the longest common prefix
        return max_prefix_length
```

Java

```java
class Solution {

    // Method to find the longest common prefix length represented in two arrays
    public int longestCommonPrefix(int[] arr1, int[] arr2) {
        // Create a HashSet to store unique prefixes
        Set<Integer> prefixes = new HashSet<>();

        // Iterate through every number in the first array
        for (int num : arr1) {
            // Loop to add all prefixes of the current number to the set
            for (int prefix = num; prefix > 0; prefix /= 10) {
                prefixes.add(prefix);
            }
        }

        // Initialize the variable `longestPrefixLength` to store the length of the longest common prefix
        int longestPrefixLength = 0;

        // Iterate through every number in the second array
        for (int num : arr2) {
            // Loop to check if any prefix of the current number exists in the set
            for (int prefix = num; prefix > 0; prefix /= 10) {
                // If a common prefix is found
                if (prefixes.contains(prefix)) {
                    // Update `longestPrefixLength` with the length of the current prefix if it's the longest found so far
                    longestPrefixLength = Math.max(longestPrefixLength, String.valueOf(prefix).length());
                    // Break the loop since we only need the longest common prefix
                    break;
                }
            }
        }

        // Return the length of the longest common prefix
        return longestPrefixLength;
    }
}
```

C++

```cpp
#include <vector>       // Required to use std::vector
#include <cmath>        // Required for log10 function
#include <unordered_set> // Required for std::unordered_set
using namespace std;   // To refrain from using `std::` prefix

class Solution {
public:
    int longestCommonPrefix(vector<int>& nums1, vector<int>& nums2) {
        // Create an unordered_set to store all the prefixes of elements in nums1
        unordered_set<int> prefixes;

        // Insert all prefixes of all numbers in nums1 into the set
        // A prefix here is obtained by repeatedly dividing the number by 10
        for (int num : nums1) {
            for (; num; num /= 10) {
                prefixes.insert(num);
            }
        }

        // Initialize `longestPrefixLength` to store the length of the longest prefix found
        int longestPrefixLength = 0;

        // Iterating through each number in nums2
        for (int num : nums2) {
            // Break down the number into prefixes and check against our set
            for (; num; num /= 10) {
                // If the current prefix is in the set, it's a common prefix
                if (prefixes.count(num)) {
                    // Logarithm base 10 of num gives us the number of digits - 1
                    longestPrefixLength = max(longestPrefixLength, (int)log10(num) + 1);
                    // once the longest prefix for current num is found, no need to look for shorter ones
                    break;
                }
            }
        }

        // Return the length of the longest common prefix
        return longestPrefixLength;
    }
};
```

TypeScript

```typescript
function longestCommonPrefix(arr1: number[], arr2: number[]): number {
    // Create a new Set to store unique digits from arr1.
    const uniqueDigits: Set<number> = new Set<number>();

    // Extract digits from each number in arr1 and add them to the Set.
    for (let num of arr1) {
        // Iterate through digits of 'num' by continuously dividing by 10.
        while (num) {
            // Add the rightmost digit to the Set and truncate 'num' to remove that digit.
            uniqueDigits.add(num % 10);
            num = Math.floor(num / 10);
        }
    }

    // Initialize 'longestPrefix' to 0, which will keep track of
    // the length of the longest prefix found that matches a digit in 'uniqueDigits'.
    let longestPrefix: number = 0;

    // Look for common digits in arr2 that exist in the 'uniqueDigits' Set.
    for (let num of arr2) {
        // Again, iterate through digits of 'num'.
        while (num) {
            // If the rightmost digit is found in 'uniqueDigits', we have a common prefix.
            if (uniqueDigits.has(num % 10)) {
                // Update 'longestPrefix' with the maximum of current value
                // and the current number 'num' length.
                longestPrefix = Math.max(longestPrefix, Math.floor(Math.log10(num)) + 1);

                // Truncate 'num' to remove its rightmost digit.
                num = Math.floor(num / 10);
            }
        }
    }

    // Return the length of the longest common prefix.
    return longestPrefix;
}


class Solution:
    def longestCommonPrefix(self, nums1: List[int], nums2: List[int]) -> int:
        # Initialize an empty set to store unique prefixes
        prefixes = set()

        # Loop through each number in the first list (nums1)
        for num in nums1:
            # Generate prefixes by continuously dividing the number by 10
            while num:
                # Add the current prefix to the set
                prefixes.add(num)
                # Update `num` by removing the last digit
                num //= 10

        # Initialize `max_prefix_length` to keep track of the longest prefix length
        max_prefix_length = 0

        # Loop through each number in the second list (nums2)
        for num in nums2:
            # Again, generate prefixes by continuously dividing the number by 10
            while num:
                # If the current prefix is in `prefixes` set, it is a common prefix
                if num in prefixes:
                    # Update `max_prefix_length` if this prefix is longer than the current max
                    max_prefix_length = max(max_prefix_length, len(str(num)))
                    # Break the loop as we've found the longest prefix for this number
                    break
                # Update `num` by removing the last digit
                num //= 10

        # Return the length of the longest common prefix
        return max_prefix_length
```

Time and Space Complexity

The time complexity of the algorithm is $O(m * \log M + n * \log N)$, where m is the length of `arr1`, n is the length of `arr2`, M is the maximum value in `arr1`, and N is the maximum value in `arr2`. This is because for each element in `arr1` and `arr2`, the algorithm potentially divides the element by 10 until it reaches 0, which occurs in $O(\log M)$ or $O(\log N)$ time for each element respectively.

The space complexity of the algorithm is $O(m * \log M)$ because the algorithm stores each prefix of the numbers from `arr1` in a set. The number of prefixes for a number with a logarithmic number of digits is also logarithmic, so each number contributes $O(\log M)$ space, and there are m numbers in `arr1`.