3042. Count Prefix and Suffix Pairs I

Easy Trie Array String String Matching Hash Function Rolling Hash

Problem Description

In this problem, we're given an array of 0-indexed strings and we need to count the number of pairs of indices `(i, j)` for which the string at index `i` is both a prefix and a suffix of the string at index `j`, and `i` must be less than `j`. A prefix of a string is a leading substring of the original string, while a suffix is a trailing substring.

For clarity, let's consider what constitutes a prefix and a suffix:

- A prefix of a string is the start of that string. For example, "cat" is a prefix of "cater".
- A suffix is the end of the string. For example, "ter" is a suffix of "cater".

Therefore, if a given string is both a prefix and a suffix of another, it appears at both the beginning and the end of the other string. The function `isPrefixAndSuffix(str1, str2)` is supposed to return true if `str1` is both a prefix and suffix of `str2`, otherwise false.

The task is to count how many such distinct index pairs `(i, j)` exist in the given array where `i < j`.

Intuition

To solve this problem, we need to look at every possible pair of strings where the first string could be a potential prefix and suffix for the second string. We can do this through a nested loop where we iterate over each string and compare it with every other string that comes after it in the array.

When comparing the strings, we use two important string operations: `startswith()` and `endswith()`. These methods return true if the first string is a prefix or suffix of the second string, respectively:

1. The method `startswith(s)` will check if the string under consideration starts with the substring `s`.
2. Likewise, `endswith(s)` will verify if the string ends with the substring `s`.

By ensuring that both these conditions are true for a pair of strings, we confirm that the first string is both a prefix and a suffix of the second one.

We only consider pairs where the first string index is less than the second (`i < j`), to comply with the problem statement. For each such valid pair, we increment our answer. This ensures that we count all the unique pairs that satisfy the condition. The summed total gives us the desired output.

Solution Approach

The implementation of the solution utilizes a straightforward brute force approach, where we simply iterate through every possible pair of strings and check if one string is both a prefix and a suffix of the other.

The algorithm can be described as follows:

1. Initialize a counter (let's call it `ans`) and set it to 0. This will hold the count of valid pairs `(i, j)` we find.
2. Use nested loops to iterate through the array of strings:
   - The outer loop will go through each string `s` in the array, starting from the first element up to the second-to-last. We use `enumerate` to get the index `i` alongside the string `s`.
   - The inner loop will iterate through the remaining elements in the array, starting from the index `i + 1` and continuing to the end of the array. These will be the strings `t` against which we will compare `s`.
3. For each pair of strings `(i, t)`, where `s` is from the outer loop and `t` is from the inner loop:
   - We check if string `t` starts with `s` using `t.startswith(s)`.
   - Simultaneously, we check if string `t` ends with `s` using `t.endswith(s)`.
   - If both conditions are `true`, meaning `s` is both a prefix and suffix of `t`, increment `ans` by 1.
4. After the loops complete, return the count `ans` which now contains the number of index pairs `(i, j)` where `isPrefixAndSuffix(words[i], words[j])` is `true`.

There are no advanced data structures utilized in this solution; it's a simple application of nested iteration over the given array and string comparison methods. The efficiency of the algorithm is not optimized; it operates with a time complexity of O(n^2 * m) where `n` is the length of the `words` array, and `m` is the average string length. This is because for every pair of strings, we potentially check each character up to the length of the shorter string twice (once for prefix, once for suffix).

Even though the solution is not the most optimal, it works well for small datasets and serves as a clear example of how to implement such a count conditionally on a collection of strings.

Example Walkthrough

Let's consider a simple example to illustrate the solution approach. Suppose we are given the following array of strings:

```
words = ["one", "oneon", "neon", "oneonne"]
```

We want to find the count of pairs of indices `(i, j)` such that the string at index `i` is both a prefix and a suffix of the string at index `j`, with `i < j`.

Here's how we apply the solution step by step:

1. Initialize `ans` to 0.
2. Iterate over the array to consider every potential string `s`:
   - At `i = 0`, `s = "one"`.
     - Now we compare `s` with every other string `t` with index greater than `i`.
     - At `j = 1`, `t = "oneon"`. We find that `t.startswith(s)` is `True` but `t.endswith(s)` is `False`. Since both conditions are not met, we do not increment `ans`.
     - At `j = 2`, `t = "neon"`. Neither `t.startswith(s)` nor `t.endswith(s)` are `True`, so we continue.
     - At `j = 3`, `t = "oneonne"`. Both `t.startswith(s)` and `t.endswith(s)` are `True`. So, we increment `ans` by 1. Now, `ans = 1`.
   - Move to `i = 1`, `s = "oneon"`. Repeat the steps, but there are no strings after `s` that satisfy the conditions.
   - Continue to `i = 2`, `s = "neon"`. There's only one string after this, and it doesn't satisfy the conditions.
3. No further pairs can be considered because `i` should be less than `j`, and we have exhausted all possibilities.
4. Finally, return the count `ans`, which in this case is `1`.

Therefore, there is only 1 valid pair `(0, 3)` where the first string is both a prefix and a suffix of the second string, and `i` is less than `j`.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def count_prefix_suffix_pairs(self, words: List[str]) -> int:
        # Initialize a variable to count the pairs
        pair_count = 0

        # Iterate through each word in the list
        for i, source_word in enumerate(words):
            # Compare with every other word in the list that comes after the current word
            for target_word in words[i + 1:]:
                # Increase the count if target_word starts and ends with source_word
                pair_count += target_word.endswith(source_word) and target_word.startswith(source_word)

        # Return the total count of pairs
        return pair_count
```

Java

```java
class Solution {
    // Method to count the number of pairs where one word is both a prefix and a suffix of another word.
    public int countPrefixSuffixPairs(String[] words) {
        int pairCount = 0; // Initialize counter for pairs to 0.
        int wordCount = words.length; // Store the length of the words array.

        // Iterate over all words in the array using two nested loops to consider pairs.
        for (int i = 0; i < wordCount; ++i) {
            String currentWord = words[i]; // The current word for prefix/suffix checking

            // Iterate over the words following the current word to avoid duplicate pairs.
            for (int j = i + 1; j < wordCount; ++j) {
                String comparisonWord = words[j]; // Word to compare with the current word

                // Check if the comparison word starts with and ends with the current word.
                if (comparisonWord.startsWith(currentWord) && comparisonWord.endsWith(currentWord)) {
                    pairCount++; // Increment the number of valid pairs if conditions are met.
                }
            }
        }

        return pairCount; // Return the final count of valid pairs.
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to count the number of pairs of strings in the given vector 'words'
    // where one string is a prefix as well as a suffix of the other string.
    int countPrefixSuffixPairs(vector<string>& words) {
        int count = 0;              // Initialize the count of valid pairs
        int numWords = words.size();  // Get the number of words in the vector
        // Iterate over each word in the vector as the potential prefix/suffix
        for (int i = 0; i < numWords; ++i) {
            string prefixSuffix = words[i];
            // Iterate over the remaining words in the vector to find a match
            for (int j = i + 1; j < numWords; ++j) {
                string candidate = words[j];
                // Check that the word at index i is a prefix of the word at index j
                if (candidate.find(prefixSuffix) == 0 &&
                    // Additionally check that it is also a suffix of the word at index j
                    candidate.rfind(prefixSuffix) == candidate.length() - prefixSuffix.length()) {
                    count++;    // If both prefix and suffix conditions are satisfied, increment count
                }
            }
        }
        return count; // Return the total count of valid prefix and suffix pairs
    }
};
```

TypeScript

```typescript
// Counts the number of pairs where one string in the array is both a prefix and a suffix of another string.
// @param {string[]} words - An array of words to check for prefix-suffix pairs.
// @returns {number} The count of prefix-suffix pairs found in the array.
function countPrefixSuffixPairs(words: string[]): number {
    // Initialize a variable to keep track of the count of pairs.
    let pairCount = 0;

    // Loop through each word in the array.
    for (let i = 0; i < words.length; ++i) {
        const word = words[i];

        // Loop through the remaining words in the array starting from the next index.
        for (const otherWord of words.slice(i + 1)) {
            // Check if the current word is both a prefix and a suffix of the other word.
            if (otherWord.startsWith(word) && otherWord.endsWith(word)) {
                // Increment the count of pairs if the condition is met.
                ++pairCount;
            }
        }
    }

    // Return the total count of prefix-suffix pairs.
    return pairCount;
}
```

```python
from typing import List

class Solution:
    def count_prefix_suffix_pairs(self, words: List[str]) -> int:
        # Initialize a variable to count the pairs
        pair_count = 0

        # Iterate through each word in the list
        for i, source_word in enumerate(words):
            # Compare with every other word in the list that comes after the current word
            for target_word in words[i + 1:]:
                # Increase the count if target_word starts and ends with source_word
                pair_count += target_word.endswith(source_word) and target_word.startswith(source_word)

        # Return the total count of pairs
        return pair_count
```

Time and Space Complexity

Time Complexity

The provided function iterates over each word using a nested loop structure. The outer loop runs `n` times where `n` is the length of the `words` list. For each iteration of the outer loop, the inner loop runs `n - i - 1` times, where `i` is the current index of the outer loop. There, each comparison between two strings, for `startswith` and `endswith`, can take up to `m` operations where `m` is the maximum length of the strings in the `words` list. Hence, the worst-case time complexity is `O(n^2 * m)`.

Space Complexity

The function mainly uses constant extra space, only storing the `ans` variable and loop indices. It operates directly on the input and does not allocate additional space that depends on the input size. Therefore, the space complexity is `O(1)` as it remains constant regardless of the input size.