**3042. Count Prefix and Suffix Pairs I**

**Hint You are given a 0-indexed string array words.**

**Let's define a boolean function isPrefixAndSuffix that takes two strings, str1 and str2: isPrefixAndSuffix(str1, str2) returns true if str1 is both a prefix and a suffix of str2, and false otherwise.**

**For example, isPrefixAndSuffix("aba", "ababa") is true because "aba" is a prefix of "ababa" and also a suffix, but isPrefixAndSuffix("abc", "abcd") is false. Return an integer denoting the number of index pairs (i, j) such that i < j, and isPrefixAndSuffix(words[i], words[j]) is true.**

**Example 1: Input: words = ["a","aba","ababa","aa"] Output: 4 Explanation: In this example, the counted index pairs are: i = 0 and j = 1 because isPrefixAndSuffix("a", "aba") is true. i = 0 and j = 2 because isPrefixAndSuffix("a", "ababa") is true. i = 0 and j = 3 because isPrefixAndSuffix("a", "aa") is true. i = 1 and j = 2 because isPrefixAndSuffix("aba", "ababa") is true. Therefore, the answer is 4. Example 2: Input: words = ["pa","papa","ma","mama"] Output: 2 Explanation: In this example, the counted index pairs are: i = 0 and j = 1 because isPrefixAndSuffix("pa", "papa") is true. i = 2 and j = 3 because isPrefixAndSuffix("ma", "mama") is true. Therefore, the answer is 2. Example 3: Input: words = ["abab","ab"] Output: 0 Explanation: In this example, the only valid index pair is i = 0 and j = 1, and isPrefixAndSuffix("abab", "ab") is false. Therefore, the answer is 0. Constraints: 1 <= words.length <= 50 1 <= words[i].length <= 10 words[i] consists only of lowercase English letters.**

**To solve this problem, we can follow a straightforward approach by iterating through all possible pairs of indices (i,j)(i, j)(i,j) in the words array and checking if the word at index iii is both a prefix and a suffix of the word at index jjj. Here's a step-by-step breakdown:**

**Steps:**

1. **Define the Helper Function**:
   * Write a helper function isPrefixAndSuffix(str1, str2) that checks if str1 is both a prefix and a suffix of str2.
2. **Iterate Through All Pairs**:
   * Loop over all indices iii and jjj where i<ji < ji<j, and use the helper function to check the condition.
3. **Count Valid Pairs**:
   * Maintain a counter to keep track of valid (i,j)(i, j)(i,j) pairs where isPrefixAndSuffix returns true.
4. **Return the Result**:
   * Return the total count.

**Implementation in Python:**

def countPrefixAndSuffixPairs(words):

# Helper function to check prefix and suffix condition

def isPrefixAndSuffix(str1, str2):

return str2.startswith(str1) and str2.endswith(str1)

n = len(words)

count = 0

# Check all pairs (i, j) with i < j

for i in range(n):

for j in range(i + 1, n):

if isPrefixAndSuffix(words[i], words[j]):

count += 1

return count

# Example usage:

words1 = ["a", "aba", "ababa", "aa"]

words2 = ["pa", "papa", "ma", "mama"]

words3 = ["abab", "ab"]

print(countPrefixAndSuffixPairs(words1)) # Output: 4

print(countPrefixAndSuffixPairs(words2)) # Output: 2

print(countPrefixAndSuffixPairs(words3)) # Output: 0

**Explanation of the Code:**

1. **Helper Function**:
   * str2.startswith(str1) checks if str1 is a prefix of str2.
   * str2.endswith(str1) checks if str1 is a suffix of str2.
2. **Nested Loops**:
   * The outer loop iterates over the first word index iii.
   * The inner loop iterates over the second word index j>ij > ij>i, ensuring we only check pairs where i<ji < ji<j.
3. **Counting Pairs**:
   * For each valid pair, the counter count is incremented.
4. **Efficiency**:
   * The solution works efficiently for the given constraints (n≤50n \leq 50n≤50, word length ≤10\leq 10≤10).

**Time Complexity:**

* **Helper Function**: O(k)O(k)O(k), where kkk is the length of str2.
* **Nested Loops**: O(n2)O(n^2)O(n2), where nnn is the length of words.
* Overall Complexity: O(n2⋅k)O(n^2 \cdot k)O(n2⋅k), which is acceptable given the constraints.

**Examples:**

**Input:**

python

Copy code

words = ["a", "aba", "ababa", "aa"]

**Output**:

Copy code

4

**Input:**

python

Copy code

words = ["pa", "papa", "ma", "mama"]

**Output**:

Copy code

2

**Input:**

python

Copy code

words = ["abab", "ab"]

**Output**:

java simpler code

import java.util.\*;

public class PrefixAndSuffixPairs {

public static int countPrefixAndSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

// Iterate over all pairs (i, j) where i < j

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// Check if words[i] is both prefix and suffix of words[j]

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

// Helper function to check prefix and suffix condition

private static boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

public static void main(String[] args) {

String[] words1 = {"a", "aba", "ababa", "aa"};

String[] words2 = {"pa", "papa", "ma", "mama"};

String[] words3 = {"abab", "ab"};

System.out.println(countPrefixAndSuffixPairs(words1)); // Output: 4

System.out.println(countPrefixAndSuffixPairs(words2)); // Output: 2

System.out.println(countPrefixAndSuffixPairs(words3)); // Output: 0

}

}

**Explanation of the Code:**

1. **Helper Function**:
   * isPrefixAndSuffix checks if one string is both a prefix and suffix of another using startsWith and endsWith.
2. **Nested Loops**:
   * Outer loop iterates through the array for the first string.
   * Inner loop iterates through the array for the second string, starting from the index after the first string.
3. **Counter**:
   * count keeps track of valid pairs where the condition holds.

**Example Input and Output:**

**Input:**

String[] words = {"a", "aba", "ababa", "aa"};

**Output:**

Compile Error Line 7: error: cannot find symbol [in \_\_Driver\_\_.java] int ret = new Solution().countPrefixSuffixPairs(param\_1); ^ symbol: class Solution location: class \_\_DriverSolution\_\_

The error occurs because the class containing the method countPrefixSuffixPairs is not named Solution. In many coding platforms, the class should be named Solution to match their requirements.

Here is the corrected Java code with the appropriate class name:

java

public class Solution {

public int countPrefixAndSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

// Iterate over all pairs (i, j) where i < j

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// Check if words[i] is both prefix and suffix of words[j]

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

// Helper function to check prefix and suffix condition

private boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

}

**Testing the Code**

If you're using a coding platform or an IDE, you can write a test case in a main method as follows:

java

Copy code

public class Main {

public static void main(String[] args) {

Solution solution = new Solution();

String[] words1 = {"a", "aba", "ababa", "aa"};

String[] words2 = {"pa", "papa", "ma", "mama"};

String[] words3 = {"abab", "ab"};

System.out.println(solution.countPrefixAndSuffixPairs(words1)); // Output: 4

System.out.println(solution.countPrefixAndSuffixPairs(words2)); // Output: 2

System.out.println(solution.countPrefixAndSuffixPairs(words3)); // Output: 0

}

}

**Key Changes:**

1. **Class Name**:
   * Changed the class name to Solution to match coding platform conventions.
2. **Testing**:
   * Added a Main class for testing purposes if you want to compile and run locally. On coding platforms, the test cases are usually handled automatically.

Compile Error Line 1: error: class Solution is public, should be declared in a file named Solution.java [in Main.java] public class Solution { ^ Line 7: error: cannot find symbol [in \_\_Driver\_\_.java] int ret = new Solution().countPrefixSuffixPairs(param\_1); ^ symbol: method countPrefixSuffixPairs(String[]) location: class Solution 2 errors

The error occurs because the class Solution is public and is not in a file named Solution.java. In Java, if a class is declared as public, the file name must match the class name.

Additionally, the method name is incorrect in the test driver. The method is defined as countPrefixAndSuffixPairs, but the driver is calling countPrefixSuffixPairs.

Here's how you can fix it:

**Corrected Code:**

1. **Save the Class in Solution.java**: Save the Solution class in a file named Solution.java.

java

Copy code

public class Solution {

public int countPrefixAndSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

// Iterate over all pairs (i, j) where i < j

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// Check if words[i] is both prefix and suffix of words[j]

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

// Helper function to check prefix and suffix condition

private boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

}

1. **Test the Code in Main.java**: Save the following test code in a separate file named Main.java.

java

Copy code

public class Main {

public static void main(String[] args) {

Solution solution = new Solution();

String[] words1 = {"a", "aba", "ababa", "aa"};

String[] words2 = {"pa", "papa", "ma", "mama"};

String[] words3 = {"abab", "ab"};

System.out.println(solution.countPrefixAndSuffixPairs(words1)); // Output: 4

System.out.println(solution.countPrefixAndSuffixPairs(words2)); // Output: 2

System.out.println(solution.countPrefixAndSuffixPairs(words3)); // Output: 0

}

}

**Key Points:**

1. **File Names**:
   * Save Solution in Solution.java.
   * Save the test driver in a separate file (e.g., Main.java).
2. **Method Name Consistency**:
   * Ensure the method name matches between the implementation (countPrefixAndSuffixPairs) and the test driver.
3. **Compilation**:
   * Compile both files together:

bash

Copy code

javac Solution.java Main.java

* + Run the program:

bash

java Main

If you're using a coding platform, ensure that the class Solution and its method name match the requirements specified in the problem statement. Replace Main.java with the platform's testing framework if necessary.

Compile Error Line 15: error: class Solution is public, should be declared in a file named Solution.java [in Main.java] public class Solution { ^ Line 7: error: cannot find symbol [in \_\_Driver\_\_.java] int ret = new Solution().countPrefixSuffixPairs(param\_1); ^ symbol: method countPrefixSuffixPairs(String[]) location: class Solution 2 errors

On LeetCode, the Solution class must be written in a single file without making it public, as the platform does not require file names to match class names. Here's the corrected code for LeetCode:

java

class Solution {

public int countPrefixAndSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

// Iterate over all pairs (i, j) where i < j

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// Check if words[i] is both prefix and suffix of words[j]

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

// Helper function to check prefix and suffix condition

private boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

}

**Explanation:**

1. **No public Modifier**:
   * The Solution class does not need the public keyword on LeetCode since the platform does not enforce file naming rules.
2. **Method Naming**:
   * Ensure the method name matches exactly with what is specified in the problem description.

**Steps to Submit:**

1. Copy the above code.
2. Paste it into the code editor on LeetCode.
3. Click "Run Code" to verify correctness.
4. Click "Submit" to test against all cases.

Line 7: error: cannot find symbol [in \_\_Driver\_\_.java] int ret = new Solution().countPrefixSuffixPairs(param\_1); ^ symbol: method countPrefixSuffixPairs(String[]) location: class Solution

The error suggests that the method name being called in the test environment (countPrefixSuffixPairs) does not match the method name defined in the Solution class. Ensure that the method name is exactly as expected by the LeetCode platform.

The expected method name in the Solution class for this problem is likely countPrefixAndSuffixPairs. Update the method name to match:

**Correct Code for LeetCode:**

java

Copy code

class Solution {

public int countPrefixSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

// Iterate over all pairs (i, j) where i < j

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// Check if words[i] is both prefix and suffix of words[j]

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

// Helper function to check prefix and suffix condition

private boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

}

**Key Fix:**

* Changed the method name from countPrefixAndSuffixPairs to countPrefixSuffixPairs to match the expected name in the driver code (new Solution().countPrefixSuffixPairs).

**Steps to Verify:**

1. Copy the code above into the LeetCode editor.
2. Click "Run Code" to verify with sample test cases.
3. Click "Submit" to test with all test cases.

If the issue persists, double-check the problem description on LeetCode to ensure the method signature aligns with the expected method name and parameters.

**Explaination of above code**

**Code Breakdown:**

**1. Class Declaration**

java

class Solution {

* This is the class Solution required by LeetCode. Since it is used in their platform-specific driver, it doesn't need the public modifier.

**2. Main Method: countPrefixSuffixPairs**

java

Copy code

public int countPrefixSuffixPairs(String[] words) {

int count = 0;

int n = words.length;

* **Purpose**: Counts the number of valid (i, j) pairs where words[i] is both a prefix and suffix of words[j].
* count: Keeps track of the number of valid pairs.
* n: The total number of words in the input array words.

**3. Nested Loops to Check All Pairs**

java

Copy code

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

* **Outer Loop (i)**: Loops through each word in words.
* **Inner Loop (j)**: Loops through every word that comes after i (i < j), ensuring we only consider unique pairs.

This combination ensures that all unique (i, j) pairs are checked.

**4. Check the isPrefixAndSuffix Condition**

java

Copy code

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

* For each (i, j) pair, it calls the helper function isPrefixAndSuffix to check if words[i] is both:
  + A prefix of words[j].
  + A suffix of words[j].
* If true, increment the count.

**5. Return the Result**

java

return count;

* After all pairs are checked, return the total count of valid (i, j) pairs.

**6. Helper Function: isPrefixAndSuffix**

java

private boolean isPrefixAndSuffix(String str1, String str2) {

return str2.startsWith(str1) && str2.endsWith(str1);

}

* **Purpose**: Checks if str1 is both a prefix and suffix of str2.
* **Logic**:
  + str2.startsWith(str1): Checks if str2 starts with str1.
  + str2.endsWith(str1): Checks if str2 ends with str1.
* Returns true if both conditions are met.

**Example Walkthrough:**

**Input: words = ["a", "aba", "ababa", "aa"]**

* Outer loop (i) runs from 0 to 3.
* Inner loop (j) runs from i + 1 to 3.

**Steps:**

1. **Pair (0, 1): ("a", "aba")**
   * "aba".startsWith("a") → true.
   * "aba".endsWith("a") → true.
   * **Count**: 1.
2. **Pair (0, 2): ("a", "ababa")**
   * "ababa".startsWith("a") → true.
   * "ababa".endsWith("a") → true.
   * **Count**: 2.
3. **Pair (0, 3): ("a", "aa")**
   * "aa".startsWith("a") → true.
   * "aa".endsWith("a") → true.
   * **Count**: 3.
4. **Pair (1, 2): ("aba", "ababa")**
   * "ababa".startsWith("aba") → true.
   * "ababa".endsWith("aba") → true.
   * **Count**: 4.

**Complexity Analysis:**

**1. Time Complexity:**

* Outer loop runs n times.
* Inner loop runs approximately n/2 times on average.
* **Total**: O(n2)O(n^2)O(n2), where nnn is the length of the words array.
* The helper function isPrefixAndSuffix runs in O(k)O(k)O(k), where kkk is the average length of the strings.

Overall: O(n2⋅k)O(n^2 \cdot k)O(n2⋅k).

**2. Space Complexity:**

* The solution uses only a few variables (count, n, etc.), so **space complexity is O(1)O(1)O(1)**.

**Output for Example:**

**Input:** ["a", "aba", "ababa", "aa"]  
**Output:** 4

This corresponds to the pairs (0, 1), (0, 2), (0, 3), (1, 2) being valid.

4o

**You said:**

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. } } compare above code with it

Let’s compare the code with the previous code step-by-step. Both implementations solve the same problem but have slight differences in how they are written or structured.

**Similarities:**

1. **Purpose**:
   * Both codes aim to count the number of (i, j) pairs where words[i] is both a prefix and suffix of words[j].
2. **Logic**:
   * Both use nested loops to iterate over all valid (i, j) pairs.
   * Both use startsWith and endsWith methods to check if the conditions are met.
3. **Output**:
   * Both codes ultimately return the pairCount, which is the total number of valid pairs.

**Differences:**

| **Aspect** | **Code Above** | **Previous Code** |
| --- | --- | --- |
| **Variable Names** | Uses pairCount, currentWord, and comparisonWord for clarity. | Uses generic names like count, words[i], and words[j] without additional variable assignments. |
| **Readability** | Declares and assigns variables (currentWord and comparisonWord) for better readability. | Directly accesses array elements (words[i] and words[j]) without intermediate variable assignments. |
| **Efficiency** | Functionally the same, but variable assignment in the above code might slightly increase overhead. | Direct access to elements can be slightly faster but less readable. |
| **Clarity of Comments** | Includes comments explaining each step of the process in detail. | Comments were concise but sufficient for understanding the logic. |

**Which One is Better?**

**Above Code (With Variables):**

* **Pros**:
  + Easier to read for someone who is new to programming.
  + Variable names (currentWord and comparisonWord) make it clear what each represents.
  + Detailed comments improve understanding.
* **Cons**:
  + Assigning currentWord and comparisonWord might add a tiny overhead.
  + Slightly longer code.

**Previous Code (Without Variables):**

* **Pros**:
  + Slightly shorter and more direct.
  + Skips redundant assignments, making it more concise.
* **Cons**:
  + Less beginner-friendly as it uses array indexing directly.
  + May be harder to follow for someone unfamiliar with nested loops.

**Recommendation:**

* If clarity and readability are priorities (e.g., for teaching or reviewing), **use the above code** with well-named variables and detailed comments.
* If you are optimizing for performance and are comfortable with more concise code, **the previous code** is sufficient.

Both approaches are functionally correct and achieve the same result. The choice depends on your preference for readability versus brevity.

Top of Form