Problem 2135: Count Words Obtained After Adding a Letter

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given two

0-indexed

arrays of strings

startWords

and

targetWords

. Each string consists of

lowercase English letters

only.

For each string in

targetWords

, check if it is possible to choose a string from

startWords

and perform a

conversion operation

on it to be equal to that from

targetWords

.

The

conversion operation

is described in the following two steps:

Append

any lowercase letter that is

not present

in the string to its end.

For example, if the string is

"abc"

, the letters

'd'

,

'e'

, or

'y'

can be added to it, but not

'a'

. If

'd'

is added, the resulting string will be

"abcd"

.

Rearrange

the letters of the new string in

any

arbitrary order.

For example,

"abcd"

can be rearranged to

"acbd"

,

"bacd"

,

"cbda"

, and so on. Note that it can also be rearranged to

"abcd"

itself.

Return

the

number of strings

in

targetWords

that can be obtained by performing the operations on

any

string of

startWords

.

Note

that you will only be verifying if the string in

targetWords

can be obtained from a string in

startWords

by performing the operations. The strings in

startWords

do not

actually change during this process.

Example 1:

Input:

startWords = ["ant","act","tack"], targetWords = ["tack","act","acti"]

Output:

2

Explanation:

- In order to form targetWords[0] = "tack", we use startWords[1] = "act", append 'k' to it, and rearrange "actk" to "tack". - There is no string in startWords that can be used to obtain targetWords[1] = "act". Note that "act" does exist in startWords, but we

must

append one letter to the string before rearranging it. - In order to form targetWords[2] = "acti", we use startWords[1] = "act", append 'i' to it, and rearrange "acti" to "acti" itself.

Example 2:

Input:

startWords = ["ab","a"], targetWords = ["abc","abcd"]

Output:

1

Explanation:

- In order to form targetWords[0] = "abc", we use startWords[0] = "ab", add 'c' to it, and rearrange it to "abc". - There is no string in startWords that can be used to obtain targetWords[1] = "abcd".

Constraints:

1 <= startWords.length, targetWords.length <= 5 * 10

4

1 <= startWords[i].length, targetWords[j].length <= 26

Each string of

startWords

and

targetWords

consists of lowercase English letters only.

No letter occurs more than once in any string of

startWords

or

targetWords

.

Code Snippets

C++:

```
class Solution {
public:
```

```
int wordCount(vector<string>& startWords, vector<string>& targetWords) {

}
};
```

Java:

```
class Solution {
public int wordCount(String[] startWords, String[] targetWords) {

}
}
```

Python3:

```
class Solution:
def wordCount(self, startWords: List[str], targetWords: List[str]) -> int:
```

Python:

```
class Solution(object):
def wordCount(self, startWords, targetWords):
"""
:type startWords: List[str]
:type targetWords: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} startWords
* @param {string[]} targetWords
* @return {number}
*/
var wordCount = function(startWords, targetWords) {

};
```

TypeScript:

```
function wordCount(startWords: string[], targetWords: string[]): number {

};
```

C#:

```
public class Solution {
public int WordCount(string[] startWords, string[] targetWords) {

}
}
```

C:

```
int wordCount(char** startWords, int startWordsSize, char** targetWords, int
targetWordsSize) {

}
```

Go:

```
func wordCount(startWords []string, targetWords []string) int {

}
```

Kotlin:

```
class Solution {
fun wordCount(startWords: Array<String>, targetWords: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func wordCount(_ startWords: [String], _ targetWords: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn word_count(start_words: Vec<String>, target_words: Vec<String>) -> i32
{

}
}
```

Ruby:

```
# @param {String[]} start_words
# @param {String[]} target_words
# @return {Integer}
def word_count(start_words, target_words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $startWords
* @param String[] $targetWords
* @return Integer
*/
function wordCount($startWords, $targetWords) {

}
}
```

Dart:

```
class Solution {
int wordCount(List<String> startWords, List<String> targetWords) {

}
}
```

Scala:

```
object Solution {
def wordCount(startWords: Array[String], targetWords: Array[String]): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec word_count(start_words :: [String.t], target_words :: [String.t]) ::
integer
def word_count(start_words, target_words) do

end
end
```

Erlang:

```
-spec word_count(StartWords :: [unicode:unicode_binary()], TargetWords ::
[unicode:unicode_binary()]) -> integer().
word_count(StartWords, TargetWords) ->
.
```

Racket:

```
(define/contract (word-count startWords targetWords)
(-> (listof string?) (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public:
int wordCount(vector<string>& startWords, vector<string>& targetWords) {

}
};
```

Java Solution:

```
/**
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int wordCount(String[] startWords, String[] targetWords) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Words Obtained After Adding a Letter
Difficulty: Medium
Tags: array, string, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def wordCount(self, startWords: List[str], targetWords: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def wordCount(self, startWords, targetWords):
"""
:type startWords: List[str]
:type targetWords: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} startWords
* @param {string[]} targetWords
* @return {number}
*/
var wordCount = function(startWords, targetWords) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
function wordCount(startWords: string[], targetWords: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int WordCount(string[] startWords, string[] targetWords) {

}
}
```

C Solution:

```
/*
* Problem: Count Words Obtained After Adding a Letter
* Difficulty: Medium
* Tags: array, string, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int wordCount(char** startWords, int startWordsSize, char** targetWords, int
targetWordsSize) {

}
```

Go Solution:

```
// Problem: Count Words Obtained After Adding a Letter
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func wordCount(startWords []string, targetWords []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun wordCount(startWords: Array<String>, targetWords: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func wordCount(_ startWords: [String], _ targetWords: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Words Obtained After Adding a Letter
// Difficulty: Medium
// Tags: array, string, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn word_count(start_words: Vec<String>, target_words: Vec<String>) -> i32
{
```

```
}
}
```

Ruby Solution:

```
# @param {String[]} start_words
# @param {String[]} target_words
# @return {Integer}
def word_count(start_words, target_words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $startWords
* @param String[] $targetWords
* @return Integer
*/
function wordCount($startWords, $targetWords) {

}
}
```

Dart Solution:

```
class Solution {
int wordCount(List<String> startWords, List<String> targetWords) {

}
}
```

Scala Solution:

```
object Solution {
def wordCount(startWords: Array[String], targetWords: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec word_count(start_words :: [String.t], target_words :: [String.t]) ::
integer
def word_count(start_words, target_words) do

end
end
```

Erlang Solution:

```
-spec word_count(StartWords :: [unicode:unicode_binary()], TargetWords ::
[unicode:unicode_binary()]) -> integer().
word_count(StartWords, TargetWords) ->
.
```

Racket Solution:

```
(define/contract (word-count startWords targetWords)
(-> (listof string?) (listof string?) exact-integer?)
)
```