

Problem 2135: Count Words Obtained After Adding a Letter

Problem Information

Difficulty: Medium

Acceptance Rate: 43.79%

Paid Only: No

Tags: Array, Hash Table, String, Bit Manipulation, Sorting

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:

1. **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"`.
2. **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 * 10⁴ * 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:
```