Problem 893: Groups of Special-Equivalent Strings

Problem Information

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

Problem Description

You are given an array of strings of the same length

words

.

In one

move

, you can swap any two even indexed characters or any two odd indexed characters of a string

words[i]

.

Two strings

words[i]

and

words[j]

are

special-equivalent

if after any number of moves,

words[i] == words[j]

.

For example,

words[i] = "zzxy"

and

words[j] = "xyzz"

are

special-equivalent

because we may make the moves

"zzxy" -> "xzzy" -> "xyzz"

.

A

group of special-equivalent strings

from

words

is a non-empty subset of words such that:

Every pair of strings in the group are special equivalent, and

The group is the largest size possible (i.e., there is not a string

words[i]

not in the group such that

words[i]

is special-equivalent to every string in the group).

Return

the number of

groups of special-equivalent strings

from

words

.

Example 1:

Input:

words = ["abcd","cdab","cbad","xyzz","zzxy","zzyx"]

Output:

3

Explanation:

One group is ["abcd", "cdab", "cbad"], since they are all pairwise special equivalent, and none of the other strings is all pairwise special equivalent to these. The other two groups are ["xyzz", "zzxy"] and ["zzyx"]. Note that in particular, "zzxy" is not special equivalent to "zzyx".

Example 2:

Input:

words = ["abc","acb","bac","bca","cab","cba"]

Output:

3

Constraints:

1 <= words.length <= 1000

1 <= words[i].length <= 20

words[i]

consist of lowercase English letters.

All the strings are of the same length.

Code Snippets

C++:

```
class Solution {
public:
int numSpecialEquivGroups(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public int numSpecialEquivGroups(String[] words) {

}
}
```

Python3:

```
class Solution:
def numSpecialEquivGroups(self, words: List[str]) -> int:
```

Python:

```
class Solution(object):
def numSpecialEquivGroups(self, words):
"""
:type words: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @return {number}
*/
var numSpecialEquivGroups = function(words) {

};
```

TypeScript:

```
function numSpecialEquivGroups(words: string[]): number {

};
```

C#:

```
public class Solution {
public int NumSpecialEquivGroups(string[] words) {

}
}
```

C:

```
int numSpecialEquivGroups(char** words, int wordsSize) {

}
```

Go:

```
func numSpecialEquivGroups(words []string) int {

}
```

Kotlin:

```
class Solution {
fun numSpecialEquivGroups(words: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func numSpecialEquivGroups(_ words: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_special_equiv_groups(words: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {Integer}
def num_special_equiv_groups(words)

end
```

PHP:

```
class Solution {

/**
```

```
* @param String[] $words
* @return Integer
*/
function numSpecialEquivGroups($words) {

}
}
```

Dart:

```
class Solution {
int numSpecialEquivGroups(List<String> words) {

}
}
```

Scala:

```
object Solution {
def numSpecialEquivGroups(words: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_special_equiv_groups(words :: [String.t]) :: integer
def num_special_equiv_groups(words) do

end
end
```

Erlang:

```
-spec num_special_equiv_groups(Words :: [unicode:unicode_binary()]) ->
integer().
num_special_equiv_groups(Words) ->
.
```

Racket:

```
(define/contract (num-special-equiv-groups words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Groups of Special-Equivalent Strings
* 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 numSpecialEquivGroups(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Groups of Special-Equivalent Strings
* 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 numSpecialEquivGroups(String[] words) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Groups of Special-Equivalent Strings
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 numSpecialEquivGroups(self, words: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numSpecialEquivGroups(self, words):
"""
:type words: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Groups of Special-Equivalent Strings
* 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[]} words
* @return {number}
*/
var numSpecialEquivGroups = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Groups of Special-Equivalent Strings
* 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 numSpecialEquivGroups(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Groups of Special-Equivalent Strings
* 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 NumSpecialEquivGroups(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Groups of Special-Equivalent Strings
* 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 numSpecialEquivGroups(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Groups of Special-Equivalent Strings
// 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 numSpecialEquivGroups(words []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSpecialEquivGroups(words: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSpecialEquivGroups(_ words: [String]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Groups of Special-Equivalent Strings
// 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 num_special_equiv_groups(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {Integer}
def num_special_equiv_groups(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return Integer
*/
function numSpecialEquivGroups($words) {

}
}
```

Dart Solution:

```
class Solution {
int numSpecialEquivGroups(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def numSpecialEquivGroups(words: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_special_equiv_groups(words :: [String.t]) :: integer
def num_special_equiv_groups(words) do

end
end
```

Erlang Solution:

```
-spec num_special_equiv_groups(Words :: [unicode:unicode_binary()]) ->
integer().
num_special_equiv_groups(Words) ->
.
```

Racket Solution:

```
(define/contract (num-special-equiv-groups words)
(-> (listof string?) exact-integer?)
)
```