Problem 2157: Groups of Strings

Problem Information

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

Problem Description

You are given a

0-indexed

array of strings

words

. Each string consists of

lowercase English letters

only. No letter occurs more than once in any string of

words

.

Two strings

s1

and

s2

are said to be

connected

if the set of letters of

s2

can be obtained from the set of letters of

s1

by any

one

of the following operations:

Adding exactly one letter to the set of the letters of

s1

.

Deleting exactly one letter from the set of the letters of

s1

.

Replacing exactly one letter from the set of the letters of

s1

with any letter,

including

itself.

The array

words

can be divided into one or more non-intersecting

groups

. A string belongs to a group if any

one

of the following is true:

It is connected to

at least one

other string of the group.

It is the

only

string present in the group.

Note that the strings in

words

should be grouped in such a manner that a string belonging to a group cannot be connected to a string present in any other group. It can be proved that such an arrangement is always unique.

Return

an array

ans

of size

2

where:

ans[0]

is the

maximum number

of groups

words

can be divided into, and

ans[1]

is the

size of the largest

group

.

Example 1:

Input:

words = ["a","b","ab","cde"]

Output:

[2,3]

Explanation:

- words[0] can be used to obtain words[1] (by replacing 'a' with 'b'), and words[2] (by adding 'b'). So words[0] is connected to words[1] and words[2]. - words[1] can be used to obtain words[0] (by replacing 'b' with 'a'), and words[2] (by adding 'a'). So words[1] is connected to words[0] and words[2]. - words[2] can be used to obtain words[0] (by deleting 'b'), and words[1] (by deleting 'a'). So words[2] is connected to words[0] and words[1]. - words[3] is not connected to any string in words. Thus, words can be divided into 2 groups [''a'',''b'',''ab''] and [''cde'']. The size of the largest group is 3.

Example 2:

Input:

words = [''a'',''ab'',''abc'']

Output:

[1,3]

Explanation:

- words[0] is connected to words[1]. - words[1] is connected to words[0] and words[2]. - words[2] is connected to words[1]. Since all strings are connected to each other, they should be grouped together. Thus, the size of the largest group is 3.

Constraints:

1 <= words.length <= 2 * 10

4

1 <= words[i].length <= 26

words[i]

consists of lowercase English letters only.

No letter occurs more than once in

words[i]

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* groupStrings(char** words, int wordsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun groupStrings(words: Array<String>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (group-strings words)
(-> (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] groupStrings(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Groups of Strings
Difficulty: Hard
Tags: array, string, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function groupStrings(words: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] GroupStrings(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Groups of Strings
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* groupStrings(char** words, int wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Groups of Strings
// Difficulty: Hard
// Tags: array, string, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func groupStrings(words []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun groupStrings(words: Array<String>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Groups of Strings
// Difficulty: Hard
// Tags: array, string, graph
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (group-strings words)
(-> (listof string?) (listof exact-integer?))
)
```