Problem 2306: Naming a Company

Problem Information

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

Problem Description

You are given an array of strings

ideas

that represents a list of names to be used in the process of naming a company. The process of naming a company is as follows:

Choose 2

distinct

names from

ideas

, call them

idea

A

and

idea

B

.

Swap the first letters of

idea

A

and

idea

B

with each other.

If

both

of the new names are not found in the original

ideas

, then the name

idea

A

idea

B

(the

concatenation

of

idea

A

and

idea

B

, separated by a space) is a valid company name.

Otherwise, it is not a valid name.

Return

the number of

distinct

valid names for the company

.

Example 1:

Input:

ideas = ["coffee","donuts","time","toffee"]

Output:

6

Explanation:

The following selections are valid: - ("coffee", "donuts"): The company name created is "doffee conuts". - ("donuts", "coffee"): The company name created is "conuts doffee". - ("donuts", "time"): The company name created is "tonuts dime". - ("donuts", "toffee"): The company name created is "tonuts doffee". - ("time", "donuts"): The company name created is "dime tonuts". - ("toffee", "donuts"): The company name created is "doffee tonuts". Therefore, there are a total of 6 distinct company names.

The following are some examples of invalid selections: - ("coffee", "time"): The name "toffee" formed after swapping already exists in the original array. - ("time", "toffee"): Both names are still the same after swapping and exist in the original array. - ("coffee", "toffee"): Both names formed after swapping already exist in the original array.

Example 2:

Input:

ideas = ["lack","back"]

Output:

0

Explanation:

There are no valid selections. Therefore, 0 is returned.

Constraints:

2 <= ideas.length <= 5 * 10

4

1 <= ideas[i].length <= 10

ideas[i]

consists of lowercase English letters.

All the strings in

ideas

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
long long distinctNames(vector<string>& ideas) {

}
};
```

Java:

```
class Solution {
public long distinctNames(String[] ideas) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function distinctNames(ideas: string[]): number {

};
```

C#:

```
public class Solution {
public long DistinctNames(string[] ideas) {

}
}
```

C:

```
long long distinctNames(char** ideas, int ideasSize) {

}
```

Go:

```
func distinctNames(ideas []string) int64 {

}
```

Kotlin:

```
class Solution {
fun distinctNames(ideas: Array<String>): Long {

}
}
```

Swift:

```
class Solution {
func distinctNames(_ ideas: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn distinct_names(ideas: Vec<String>) -> i64 {

}
}
```

Ruby:

```
# @param {String[]} ideas
# @return {Integer}
def distinct_names(ideas)

end
```

PHP:

```
class Solution {

/**
* @param String[] $ideas
* @return Integer
*/
function distinctNames($ideas) {

}
}
```

Dart:

```
class Solution {
int distinctNames(List<String> ideas) {

}
```

```
}
```

Scala:

```
object Solution {
def distinctNames(ideas: Array[String]): Long = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec distinct_names(Ideas :: [unicode:unicode_binary()]) -> integer().
distinct_names(Ideas) ->
.
```

Racket:

```
(define/contract (distinct-names ideas)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
*
```

```
* 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:
long long distinctNames(vector<string>& ideas) {

}
};
```

Java Solution:

```
/**
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 long distinctNames(String[] ideas) {

}
}
```

Python3 Solution:

```
"""
Problem: Naming a Company
Difficulty: Hard
Tags: array, string, hash

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 distinctNames(self, ideas: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
*
* 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[]} ideas
* @return {number}
*/
var distinctNames = function(ideas) {

};
```

TypeScript Solution:

```
/**
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
```

```
*
* 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 distinctNames(ideas: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
*
* 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 long DistinctNames(string[] ideas) {

}
}
```

C Solution:

```
/*
* Problem: Naming a Company
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long distinctNames(char** ideas, int ideasSize) {
```

```
}
```

Go Solution:

```
// Problem: Naming a Company
// Difficulty: Hard
// Tags: array, string, hash
//
// 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 distinctNames(ideas []string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun distinctNames(ideas: Array<String>): Long {

}
}
```

Swift Solution:

```
class Solution {
func distinctNames(_ ideas: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Naming a Company
// Difficulty: Hard
// Tags: array, string, hash
//
// 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 distinct_names(ideas: Vec<String>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String[]} ideas
# @return {Integer}
def distinct_names(ideas)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $ideas
* @return Integer
*/
function distinctNames($ideas) {

}
}
```

Dart Solution:

```
class Solution {
int distinctNames(List<String> ideas) {

}
}
```

Scala Solution:

```
object Solution {
def distinctNames(ideas: Array[String]): Long = {
```

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec distinct_names(Ideas :: [unicode:unicode_binary()]) -> integer().
distinct_names(Ideas) ->
.
```

Racket Solution:

```
(define/contract (distinct-names ideas)
(-> (listof string?) exact-integer?)
)
```