Problem 804: Unique Morse Code Words

Problem Information

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

Problem Description

International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows:

'a'

maps to

".-"

,

'b'

maps to

"-..."

,

'c'

maps to

"-.-."

, and so on.

For convenience, the full table for the

26

letters of the English alphabet is given below:

[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

Given an array of strings

words

where each word can be written as a concatenation of the Morse code of each letter.

For example,

"cab"

can be written as

"-.-..--..."

, which is the concatenation of

"-.-."

,

".-"

, and

"-..."

. We will call such a concatenation the

transformation

of a word.

Return

the number of different

transformations

among all words we have

.

Example 1:

Input:

words = ["gin","zen","gig","msg"]

Output:

2

Explanation:

The transformation of each word is: "gin" -> "--...-." "zen" -> "--...-." "gig" -> "--...--." "msg" -> "--...--." There are 2 different transformations: "--...-." and "--...--.".

Example 2:

Input:

words = ["a"]

Output:

1

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 12

words[i]

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (unique-morse-representations words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Morse Code Words
* Difficulty: Easy
* 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:
int uniqueMorseRepresentations(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Morse Code Words
* Difficulty: Easy
* 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 int uniqueMorseRepresentations(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Unique Morse Code Words
Difficulty: Easy
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 uniqueMorseRepresentations(self, words: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Unique Morse Code Words
* Difficulty: Easy
* 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[]} words
* @return {number}
*/
var uniqueMorseRepresentations = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Morse Code Words
* Difficulty: Easy
```

```
* 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 uniqueMorseRepresentations(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Unique Morse Code Words
* Difficulty: Easy
* 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 int UniqueMorseRepresentations(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Unique Morse Code Words
* Difficulty: Easy
* 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
*/
```

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

}
```

Go Solution:

```
// Problem: Unique Morse Code Words
// Difficulty: Easy
// 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 uniqueMorseRepresentations(words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Unique Morse Code Words
// Difficulty: Easy
// 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 unique_morse_representations(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (unique-morse-representations words)
(-> (listof string?) exact-integer?)
)
```