Problem 3706: Maximum Distance Between Unequal Words in Array II

Problem Information

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

Problem Description

You are given a string array

words

.

Find the

maximum distance

between two

distinct

indices

i

and

j

such that:

words[i] != words[j]

, and

the distance is defined as

j - i + 1

.

Return the maximum distance among all such pairs. If no valid pair exists, return 0.

Example 1:

Input:

words = ["leetcode","leetcode","codeforces"]

Output:

3

Explanation:

In this example,

words[0]

and

words[2]

are not equal, and they have the maximum distance

2 - 0 + 1 = 3

.

Example 2:

Input:

words = ["a","b","c","a","a"]

Output:

4

Explanation:

In this example

words[1]

and

words[4]

have the largest distance of

4 - 1 + 1 = 4

.

Example 3:

Input:

words = ["z","z","z"]

Output:

0

Explanation:

In this example all the words are equal, thus the answer is 0.

Constraints:

1 <= words.length <= 10

5

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

words[i]

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maxDistance(self, words):
```

```
"""
:type words: List[str]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-distance words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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 maxDistance(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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 maxDistance(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Distance Between Unequal Words in Array II
Difficulty: Medium
Tags: array, string
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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 maxDistance = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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 maxDistance(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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 MaxDistance(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Distance Between Unequal Words in Array II
* Difficulty: Medium
* Tags: array, string
*
* 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
```

```
*/

int maxDistance(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Maximum Distance Between Unequal Words in Array II
// Difficulty: Medium
// Tags: array, string
//
// 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 maxDistance(words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Distance Between Unequal Words in Array II
// Difficulty: Medium
// Tags: array, string
```

```
//
// 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 max_distance(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-distance words)
(-> (listof string?) exact-integer?)
)
```