Problem 243: Shortest Word Distance

Problem Information

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

Problem Description

Given an array of strings

wordsDict

and two different strings that already exist in the array

word1

and

word2

, return

the shortest distance between these two words in the list

.

Example 1:

Input:

wordsDict = ["practice", "makes", "perfect", "coding", "makes"], word1 = "coding", word2 = "practice"

Output:

3

Example 2:

Input:

wordsDict = ["practice", "makes", "perfect", "coding", "makes"], word1 = "makes", word2 = "coding"

Output:

1

Constraints:

2 <= wordsDict.length <= 3 * 10

4

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

wordsDict[i]

consists of lowercase English letters.

word1

and

word2

are in

wordsDict

.

word1 != word2

Code Snippets

C++:

```
class Solution {
public:
int shortestDistance(vector<string>& wordsDict, string word1, string word2) {

}
};
```

Java:

```
class Solution {
public int shortestDistance(String[] wordsDict, String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def shortestDistance(self, wordsDict: List[str], word1: str, word2: str) ->
int:
```

Python:

```
class Solution(object):
def shortestDistance(self, wordsDict, word1, word2):
"""
:type wordsDict: List[str]
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} wordsDict
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var shortestDistance = function(wordsDict, word1, word2) {

};
```

TypeScript:

```
function shortestDistance(wordsDict: string[], word1: string, word2: string):
number {

};
```

C#:

```
public class Solution {
public int ShortestDistance(string[] wordsDict, string word1, string word2) {

}
}
```

C:

```
int shortestDistance(char** wordsDict, int wordsDictSize, char* word1, char*
word2) {

}
```

Go:

```
func shortestDistance(wordsDict []string, word1 string, word2 string) int {

}
```

Kotlin:

```
class Solution {
fun shortestDistance(wordsDict: Array<String>, word1: String, word2: String):
Int {

}
}
```

Swift:

```
class Solution {
func shortestDistance(_ wordsDict: [String], _ word1: String, _ word2:
String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_distance(words_dict: Vec<String>, word1: String, word2:
String) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words_dict
# @param {String} word1
# @param {String} word2
# @return {Integer}
def shortest_distance(words_dict, word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String[] $wordsDict
* @param String $word1
* @param String $word2
* @return Integer
*/
function shortestDistance($wordsDict, $word1, $word2) {

}
}
```

Dart:

```
class Solution {
int shortestDistance(List<String> wordsDict, String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def shortestDistance(wordsDict: Array[String], word1: String, word2: String):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_distance(words_dict :: [String.t], word1 :: String.t, word2 ::
String.t) :: integer
def shortest_distance(words_dict, word1, word2) do

end
end
```

Erlang:

```
-spec shortest_distance(WordsDict :: [unicode:unicode_binary()], Word1 ::
unicode:unicode_binary(), Word2 :: unicode:unicode_binary()) -> integer().
shortest_distance(WordsDict, Word1, Word2) ->
.
```

Racket:

```
(define/contract (shortest-distance wordsDict word1 word2)
(-> (listof string?) string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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 shortestDistance(vector<string>& wordsDict, string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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 shortestDistance(String[] wordsDict, String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Word Distance
Difficulty: Easy
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 shortestDistance(self, wordsDict: List[str], word1: str, word2: str) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shortestDistance(self, wordsDict, word1, word2):
"""
:type wordsDict: List[str]
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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[]} wordsDict
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var shortestDistance = function(wordsDict, word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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 shortestDistance(wordsDict: string[], word1: string, word2: string):
number {

};
```

C# Solution:

```
/*
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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 ShortestDistance(string[] wordsDict, string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Word Distance
* Difficulty: Easy
* 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 shortestDistance(char** wordsDict, int wordsDictSize, char* word1, char*
word2) {

}
```

Go Solution:

```
// Problem: Shortest Word Distance
// Difficulty: Easy
// 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 shortestDistance(wordsDict []string, word1 string, word2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestDistance(wordsDict: Array<String>, word1: String, word2: String):
Int {

}
}
```

Swift Solution:

```
class Solution {
func shortestDistance(_ wordsDict: [String], _ word1: String, _ word2:
String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Word Distance
// Difficulty: Easy
// 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 shortest_distance(words_dict: Vec<String>, word1: String, word2:
String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words_dict
# @param {String} word1
# @param {String} word2
# @return {Integer}
def shortest_distance(words_dict, word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $wordsDict
* @param String $word1
* @param String $word2
* @return Integer
*/
function shortestDistance($wordsDict, $word1, $word2) {

}
}
```

Dart Solution:

```
class Solution {
int shortestDistance(List<String> wordsDict, String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def shortestDistance(wordsDict: Array[String], word1: String, word2: String):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_distance(words_dict :: [String.t], word1 :: String.t, word2 ::
String.t) :: integer
def shortest_distance(words_dict, word1, word2) do

end
end
```

Erlang Solution:

```
-spec shortest_distance(WordsDict :: [unicode:unicode_binary()], Word1 ::
unicode:unicode_binary(), Word2 :: unicode:unicode_binary()) -> integer().
shortest_distance(WordsDict, Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (shortest-distance wordsDict word1 word2)
(-> (listof string?) string? string? exact-integer?)
)
```