Problem 244: Shortest Word Distance II

Problem Information

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

Problem Description

Design a data structure that will be initialized with a string array, and then it should answer queries of the shortest distance between two different strings from the array.

Implement the

WordDistance

class:

WordDistance(String[] wordsDict)

initializes the object with the strings array

wordsDict

.

int shortest(String word1, String word2)

returns the shortest distance between

word1

and

word2

in the array

wordsDict

.

Example 1:

Input

["WordDistance", "shortest", "shortest"] [[["practice", "makes", "perfect", "coding", "makes"]], ["coding", "practice"], ["makes", "coding"]]

Output

[null, 3, 1]

Explanation

WordDistance wordDistance = new WordDistance(["practice", "makes", "perfect", "coding", "makes"]); wordDistance.shortest("coding", "practice"); // return 3
wordDistance.shortest("makes", "coding"); // return 1

Constraints:

1 <= 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

At most

5000

calls will be made to

shortest

.

Code Snippets

C++:

```
class WordDistance {
public:
WordDistance(vector<string>& wordsDict) {

}

int shortest(string word1, string word2) {

}
};

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance* obj = new WordDistance(wordsDict);
* int param_1 = obj->shortest(word1,word2);
```

```
*/
```

Java:

```
class WordDistance {

public WordDistance(String[] wordsDict) {

}

public int shortest(String word1, String word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = new WordDistance(wordsDict);
* int param_1 = obj.shortest(word1,word2);
*/
```

Python3:

```
class WordDistance:

def __init__(self, wordsDict: List[str]):


def shortest(self, word1: str, word2: str) -> int:




# Your WordDistance object will be instantiated and called as such:
# obj = WordDistance(wordsDict)
# param_1 = obj.shortest(word1,word2)
```

Python:

```
class WordDistance(object):

def __init__(self, wordsDict):
"""
```

```
        :type wordsDict: List[str]
        """


    def shortest(self, word1, word2):
        """
        :type word1: str
        :type word2: str
        :rtype: int
        """



# Your WordDistance object will be instantiated and called as such:
# obj = WordDistance(wordsDict)
# param_1 = obj.shortest(word1,word2)
```

JavaScript:

```
/**
* @param {string[]} wordsDict
*/
var WordDistance = function(wordsDict) {

};

/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
WordDistance.prototype.shortest = function(word1, word2) {

};

/**
* Your WordDistance object will be instantiated and called as such:
* var obj = new WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

TypeScript:

```
class WordDistance {
constructor(wordsDict: string[]) {

}

shortest(word1: string, word2: string): number {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* var obj = new WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

C#:

```
public class WordDistance {

public WordDistance(string[] wordsDict) {

}

public int Shortest(string word1, string word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = new WordDistance(wordsDict);
* int param_1 = obj.Shortest(word1,word2);
*/
```

C:

```
typedef struct {
```

```
} WordDistance;


WordDistance* wordDistanceCreate(char** wordsDict, int wordsDictSize) {

}

int wordDistanceShortest(WordDistance* obj, char* word1, char* word2) {

}

void wordDistanceFree(WordDistance* obj) {

}

/**
* Your WordDistance struct will be instantiated and called as such:
* WordDistance* obj = wordDistanceCreate(wordsDict, wordsDictSize);
* int param_1 = wordDistanceShortest(obj, word1, word2);

* wordDistanceFree(obj);
*/
```

Go:

```
type WordDistance struct {

}


func Constructor(wordsDict []string) WordDistance {

}


func (this *WordDistance) Shortest(word1 string, word2 string) int {

}


/**
```

```
* Your WordDistance object will be instantiated and called as such:
* obj := Constructor(wordsDict);
* param_1 := obj.Shortest(word1,word2);
*/
```

Kotlin:

```
class WordDistance(wordsDict: Array<String>) {

fun shortest(word1: String, word2: String): Int {

}

}

/**
* Your WordDistance object will be instantiated and called as such:
* var obj = WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

Swift:

```
class WordDistance {

init(_ wordsDict: [String]) {

}

func shortest(_ word1: String, _ word2: String) -> Int {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* let obj = WordDistance(wordsDict)
* let ret_1: Int = obj.shortest(word1, word2)
*/
```

Rust:

```
struct WordDistance {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl WordDistance {

fn new(wordsDict: Vec<String>) -> Self {

}

fn shortest(&self, word1: String, word2: String) -> i32 {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* let obj = WordDistance::new(wordsDict);
* let ret_1: i32 = obj.shortest(word1, word2);
*/
```

Ruby:

```
class WordDistance

=begin
:type words_dict: String[]
=end
def initialize(words_dict)

end


=begin
:type word1: String
:type word2: String
:rtype: Integer
=end
```

```
def shortest(word1, word2)

end


end

# Your WordDistance object will be instantiated and called as such:
# obj = WordDistance.new(words_dict)
# param_1 = obj.shortest(word1, word2)
```

PHP:

```
class WordDistance {
/**
* @param String[] $wordsDict
*/
function __construct($wordsDict) {

}

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function shortest($word1, $word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* $obj = WordDistance($wordsDict);
* $ret_1 = $obj->shortest($word1, $word2);
*/
```

Dart:

```
class WordDistance {

WordDistance(List<String> wordsDict) {
```

```
}

int shortest(String word1, String word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = WordDistance(wordsDict);
* int param1 = obj.shortest(word1,word2);
*/
```

Scala:

```
class WordDistance(_wordsDict: Array[String]) {

def shortest(word1: String, word2: String): Int = {

}

}

/**
* Your WordDistance object will be instantiated and called as such:
* val obj = new WordDistance(wordsDict)
* val param_1 = obj.shortest(word1,word2)
*/
```

Elixir:

```
defmodule WordDistance do
@spec init_(words_dict :: [String.t]) :: any
def init_(words_dict) do

end

@spec shortest(word1 :: String.t, word2 :: String.t) :: integer
def shortest(word1, word2) do

end
```

```
end

# Your functions will be called as such:
# WordDistance.init_(words_dict)
# param_1 = WordDistance.shortest(word1, word2)

# WordDistance.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec word_distance_init_(WordsDict :: [unicode:unicode_binary()]) -> any().
word_distance_init_(WordsDict) ->
.

-spec word_distance_shortest(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
word_distance_shortest(Word1, Word2) ->
.


%% Your functions will be called as such:
%% word_distance_init_(WordsDict),
%% Param_1 = word_distance_shortest(Word1, Word2),

%% word_distance_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
(define word-distance%
(class object%
(super-new)

; words-dict : (listof string?)
(init-field
words-dict)

; shortest : string? string? -> exact-integer?
(define/public (shortest word1 word2)
)))
```

```
;; Your word-distance% object will be instantiated and called as such:
;; (define obj (new word-distance% [words-dict words-dict]))
;; (define param_1 (send obj shortest word1 word2))
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Word Distance II
* Difficulty: Medium
* 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 WordDistance {
public:
WordDistance(vector<string>& wordsDict) {

}

int shortest(string word1, string word2) {

}
};

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance* obj = new WordDistance(wordsDict);
* int param_1 = obj->shortest(word1,word2);
*/
```

Java Solution:

```
/**
* Problem: Shortest Word Distance II
* Difficulty: Medium
```

```
* 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 WordDistance {

public WordDistance(String[] wordsDict) {

}

public int shortest(String word1, String word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = new WordDistance(wordsDict);
* int param_1 = obj.shortest(word1,word2);
*/
```

Python3 Solution:

```
"""
Problem: Shortest Word Distance II
Difficulty: Medium
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 WordDistance:

def __init__(self, wordsDict: List[str]):
```

```
def shortest(self, word1: str, word2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class WordDistance(object):

def __init__(self, wordsDict):
"""
:type wordsDict: List[str]
"""



def shortest(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""




# Your WordDistance object will be instantiated and called as such:
# obj = WordDistance(wordsDict)
# param_1 = obj.shortest(word1,word2)
```

JavaScript Solution:

```
/**
* Problem: Shortest Word Distance II
* Difficulty: Medium
* 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[]} wordsDict
```

```
*/
var WordDistance = function(wordsDict) {

};

/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
WordDistance.prototype.shortest = function(word1, word2) {

};

/**
* Your WordDistance object will be instantiated and called as such:
* var obj = new WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

TypeScript Solution:

```
/**
* Problem: Shortest Word Distance II
* Difficulty: Medium
* 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 WordDistance {
constructor(wordsDict: string[]) {

}

shortest(word1: string, word2: string): number {

}
}
```

```
/**
* Your WordDistance object will be instantiated and called as such:
* var obj = new WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

C# Solution:

```
/*
* Problem: Shortest Word Distance II
* Difficulty: Medium
* 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 WordDistance {

public WordDistance(string[] wordsDict) {

}

public int Shortest(string word1, string word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = new WordDistance(wordsDict);
* int param_1 = obj.Shortest(word1,word2);
*/
```

C Solution:

```
/*
* Problem: Shortest Word Distance II
* Difficulty: Medium
```

```
* 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
*/




typedef struct {

} WordDistance;


WordDistance* wordDistanceCreate(char** wordsDict, int wordsDictSize) {

}

int wordDistanceShortest(WordDistance* obj, char* word1, char* word2) {

}

void wordDistanceFree(WordDistance* obj) {

}

/**
* Your WordDistance struct will be instantiated and called as such:
* WordDistance* obj = wordDistanceCreate(wordsDict, wordsDictSize);
* int param_1 = wordDistanceShortest(obj, word1, word2);

* wordDistanceFree(obj);
*/
```

Go Solution:

```
// Problem: Shortest Word Distance II
// Difficulty: Medium
// 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

type WordDistance struct {

}


func Constructor(wordsDict []string) WordDistance {

}


func (this *WordDistance) Shortest(word1 string, word2 string) int {

}


/**
* Your WordDistance object will be instantiated and called as such:
* obj := Constructor(wordsDict);
* param_1 := obj.Shortest(word1,word2);
*/
```

Kotlin Solution:

```
class WordDistance(wordsDict: Array<String>) {

fun shortest(word1: String, word2: String): Int {

}

}

/**
* Your WordDistance object will be instantiated and called as such:
* var obj = WordDistance(wordsDict)
* var param_1 = obj.shortest(word1,word2)
*/
```

Swift Solution:

```
class WordDistance {

init(_ wordsDict: [String]) {

}

func shortest(_ word1: String, _ word2: String) -> Int {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* let obj = WordDistance(wordsDict)
* let ret_1: Int = obj.shortest(word1, word2)
*/
```

Rust Solution:

```
// Problem: Shortest Word Distance II
// Difficulty: Medium
// 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

struct WordDistance {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl WordDistance {

fn new(wordsDict: Vec<String>) -> Self {
```

```
}

fn shortest(&self, word1: String, word2: String) -> i32 {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* let obj = WordDistance::new(wordsDict);
* let ret_1: i32 = obj.shortest(word1, word2);
*/
```

Ruby Solution:

```
class WordDistance

=begin
:type words_dict: String[]
=end
def initialize(words_dict)

end


=begin
:type word1: String
:type word2: String
:rtype: Integer
=end
def shortest(word1, word2)

end


end

# Your WordDistance object will be instantiated and called as such:
# obj = WordDistance.new(words_dict)
# param_1 = obj.shortest(word1, word2)
```

PHP Solution:

```
class WordDistance {
/**
* @param String[] $wordsDict
*/
function __construct($wordsDict) {

}

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function shortest($word1, $word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* $obj = WordDistance($wordsDict);
* $ret_1 = $obj->shortest($word1, $word2);
*/
```

Dart Solution:

```
class WordDistance {

WordDistance(List<String> wordsDict) {

}

int shortest(String word1, String word2) {

}
}

/**
* Your WordDistance object will be instantiated and called as such:
* WordDistance obj = WordDistance(wordsDict);
* int param1 = obj.shortest(word1,word2);
```

```
*/
```

Scala Solution:

```
class WordDistance(_wordsDict: Array[String]) {

def shortest(word1: String, word2: String): Int = {

}

}

/**
* Your WordDistance object will be instantiated and called as such:
* val obj = new WordDistance(wordsDict)
* val param_1 = obj.shortest(word1,word2)
*/
```

Elixir Solution:

```
defmodule WordDistance do
@spec init_(words_dict :: [String.t]) :: any
def init_(words_dict) do

end

@spec shortest(word1 :: String.t, word2 :: String.t) :: integer
def shortest(word1, word2) do

end
end

# Your functions will be called as such:
# WordDistance.init_(words_dict)
# param_1 = WordDistance.shortest(word1, word2)

# WordDistance.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec word_distance_init_(WordsDict :: [unicode:unicode_binary()]) -> any().
word_distance_init_(WordsDict) ->
.

-spec word_distance_shortest(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
word_distance_shortest(Word1, Word2) ->
.


%% Your functions will be called as such:
%% word_distance_init_(WordsDict),
%% Param_1 = word_distance_shortest(Word1, Word2),

%% word_distance_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
(define word-distance%
(class object%
(super-new)

; words-dict : (listof string?)
(init-field
words-dict)

; shortest : string? string? -> exact-integer?
(define/public (shortest word1 word2)
)))

;; Your word-distance% object will be instantiated and called as such:
;; (define obj (new word-distance% [words-dict words-dict]))
;; (define param_1 (send obj shortest word1 word2))
```