Problem 288: Unique Word Abbreviation

Problem Information

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

Problem Description

The

abbreviation

of a word is a concatenation of its first letter, the number of characters between the first and last letter, and its last letter. If a word has only two characters, then it is an

abbreviation

of itself.

For example:

dog --> d1g

because there is one letter between the first letter

'd'

and the last letter

'g'

.

internationalization --> i18n

because there are 18 letters between the first letter

'i'

and the last letter

'n'

.

it --> it

because any word with only two characters is an

abbreviation

of itself.

Implement the

ValidWordAbbr

class:

ValidWordAbbr(String[] dictionary)

Initializes the object with a

dictionary

of words.

boolean isUnique(string word)

Returns

true

if

either

of the following conditions are met (otherwise returns

false

):

There is no word in

dictionary

whose

abbreviation

is equal to

word

's

abbreviation

.

For any word in

dictionary

whose

abbreviation

is equal to

word

's

abbreviation

, that word and

word

are

the same

.

Example 1:

Input

["ValidWordAbbr", "isUnique", "isUnique", "isUnique", "isUnique", "isUnique"] [[["deer", "door", "cake", "card"]], ["dear"], ["cart"], ["cane"], ["make"], ["cake"]]

Output

[null, false, true, false, true, true]

Explanation

ValidWordAbbr validWordAbbr = new ValidWordAbbr(["deer", "door", "cake", "card"]); validWordAbbr.isUnique("dear"); // return false, dictionary word "deer" and word "dear" have the same abbreviation "d2r" but are not the same. validWordAbbr.isUnique("cart"); // return true, no words in the dictionary have the abbreviation "c2t". validWordAbbr.isUnique("cane"); // return false, dictionary word "cake" and word "cane" have the same abbreviation "c2e" but are not the same. validWordAbbr.isUnique("make"); // return true, no words in the dictionary have the abbreviation "m2e". validWordAbbr.isUnique("cake"); // return true, because "cake" is already in the dictionary and no other word in the dictionary has "c2e" abbreviation.

Constraints:

1 <= dictionary.length <= 3 * 10

4

1 <= dictionary[i].length <= 20

dictionary[i]

consists of lowercase English letters.

1 <= word.length <= 20

word

consists of lowercase English letters.

At most

5000

calls will be made to

isUnique

.

Code Snippets

C++:

```
class ValidWordAbbr {
public:
ValidWordAbbr(vector<string>& dictionary) {

}

bool isUnique(string word) {

}
};
```

```
/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr* obj = new ValidWordAbbr(dictionary);
* bool param_1 = obj->isUnique(word);
*/
```

Java:

```
class ValidWordAbbr {

public ValidWordAbbr(String[] dictionary) {

}

public boolean isUnique(String word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = new ValidWordAbbr(dictionary);
* boolean param_1 = obj.isUnique(word);
*/
```

Python3:

```
class ValidWordAbbr:

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


def isUnique(self, word: str) -> bool:



# Your ValidWordAbbr object will be instantiated and called as such:
# obj = ValidWordAbbr(dictionary)
# param_1 = obj.isUnique(word)
```

Python:

```
class ValidWordAbbr(object):

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



def isUnique(self, word):
"""
:type word: str
:rtype: bool
"""




# Your ValidWordAbbr object will be instantiated and called as such:
# obj = ValidWordAbbr(dictionary)
# param_1 = obj.isUnique(word)
```

JavaScript:

```
/**
* @param {string[]} dictionary
*/
var ValidWordAbbr = function(dictionary) {

};

/**
* @param {string} word
* @return {boolean}
*/
ValidWordAbbr.prototype.isUnique = function(word) {

};

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* var obj = new ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

TypeScript:

```
class ValidWordAbbr {
constructor(dictionary: string[]) {

}

isUnique(word: string): boolean {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* var obj = new ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

C#:

```
public class ValidWordAbbr {

public ValidWordAbbr(string[] dictionary) {

}

public bool IsUnique(string word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = new ValidWordAbbr(dictionary);
* bool param_1 = obj.IsUnique(word);
*/
```

C:

```



typedef struct {
```

```
} ValidWordAbbr;


ValidWordAbbr* validWordAbbrCreate(char** dictionary, int dictionarySize) {

}

bool validWordAbbrIsUnique(ValidWordAbbr* obj, char* word) {

}

void validWordAbbrFree(ValidWordAbbr* obj) {

}

/**
* Your ValidWordAbbr struct will be instantiated and called as such:
* ValidWordAbbr* obj = validWordAbbrCreate(dictionary, dictionarySize);
* bool param_1 = validWordAbbrIsUnique(obj, word);

* validWordAbbrFree(obj);
*/
```

Go:

```
type ValidWordAbbr struct {

}


func Constructor(dictionary []string) ValidWordAbbr {

}


func (this *ValidWordAbbr) IsUnique(word string) bool {

}


/**
```

```
* Your ValidWordAbbr object will be instantiated and called as such:
* obj := Constructor(dictionary);
* param_1 := obj.IsUnique(word);
*/
```

Kotlin:

```
class ValidWordAbbr(dictionary: Array<String>) {

fun isUnique(word: String): Boolean {

}

}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* var obj = ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

Swift:

```
class ValidWordAbbr {

init(_ dictionary: [String]) {

}

func isUnique(_ word: String) -> Bool {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* let obj = ValidWordAbbr(dictionary)
* let ret_1: Bool = obj.isUnique(word)
*/
```

Rust:

```
struct ValidWordAbbr {

}


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

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

}

fn is_unique(&self, word: String) -> bool {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* let obj = ValidWordAbbr::new(dictionary);
* let ret_1: bool = obj.is_unique(word);
*/
```

Ruby:

```
class ValidWordAbbr

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

end


=begin
:type word: String
:rtype: Boolean
=end
def is_unique(word)
```

```
end


end

# Your ValidWordAbbr object will be instantiated and called as such:
# obj = ValidWordAbbr.new(dictionary)
# param_1 = obj.is_unique(word)
```

PHP:

```
class ValidWordAbbr {
/**
* @param String[] $dictionary
*/
function __construct($dictionary) {

}

/**
* @param String $word
* @return Boolean
*/
function isUnique($word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* $obj = ValidWordAbbr($dictionary);
* $ret_1 = $obj->isUnique($word);
*/
```

Dart:

```
class ValidWordAbbr {

ValidWordAbbr(List<String> dictionary) {

}
```

```
bool isUnique(String word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = ValidWordAbbr(dictionary);
* bool param1 = obj.isUnique(word);
*/
```

Scala:

```
class ValidWordAbbr(_dictionary: Array[String]) {

def isUnique(word: String): Boolean = {

}

}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* val obj = new ValidWordAbbr(dictionary)
* val param_1 = obj.isUnique(word)
*/
```

Elixir:

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

end

@spec is_unique(word :: String.t) :: boolean
def is_unique(word) do

end
end
```

```
# Your functions will be called as such:
# ValidWordAbbr.init_(dictionary)
# param_1 = ValidWordAbbr.is_unique(word)

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

Erlang:

```
-spec valid_word_abbr_init_(Dictionary :: [unicode:unicode_binary()]) -> any().
valid_word_abbr_init_(Dictionary) ->
.

-spec valid_word_abbr_is_unique(Word :: unicode:unicode_binary()) -> boolean().
valid_word_abbr_is_unique(Word) ->
.


%% Your functions will be called as such:
%% valid_word_abbr_init_(Dictionary),
%% Param_1 = valid_word_abbr_is_unique(Word),

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

Racket:

```
(define valid-word-abbr%
(class object%
(super-new)

; dictionary : (listof string?)
(init-field
dictionary)

; is-unique : string? -> boolean?
(define/public (is-unique word)
)))

;; Your valid-word-abbr% object will be instantiated and called as such:
```

```
;; (define obj (new valid-word-abbr% [dictionary dictionary]))
;; (define param_1 (send obj is-unique word))
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Word Abbreviation
* 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 ValidWordAbbr {
public:
ValidWordAbbr(vector<string>& dictionary) {

}

bool isUnique(string word) {

}
};

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr* obj = new ValidWordAbbr(dictionary);
* bool param_1 = obj->isUnique(word);
*/
```

Java Solution:

```
/**
* Problem: Unique Word Abbreviation
* 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 ValidWordAbbr {

public ValidWordAbbr(String[] dictionary) {

}

public boolean isUnique(String word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = new ValidWordAbbr(dictionary);
* boolean param_1 = obj.isUnique(word);
*/
```

Python3 Solution:

```
"""
Problem: Unique Word Abbreviation
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 ValidWordAbbr:

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


def isUnique(self, word: str) -> bool:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class ValidWordAbbr(object):

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



def isUnique(self, word):
"""
:type word: str
:rtype: bool
"""




# Your ValidWordAbbr object will be instantiated and called as such:
# obj = ValidWordAbbr(dictionary)
# param_1 = obj.isUnique(word)
```

JavaScript Solution:

```
/**
* Problem: Unique Word Abbreviation
* 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[]} dictionary
*/
var ValidWordAbbr = function(dictionary) {
```

```
};

/**
* @param {string} word
* @return {boolean}
*/
ValidWordAbbr.prototype.isUnique = function(word) {

};

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* var obj = new ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

TypeScript Solution:

```
/**
* Problem: Unique Word Abbreviation
* 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 ValidWordAbbr {
constructor(dictionary: string[]) {

}

isUnique(word: string): boolean {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
```

```
* var obj = new ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

C# Solution:

```
/*
* Problem: Unique Word Abbreviation
* 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 ValidWordAbbr {

public ValidWordAbbr(string[] dictionary) {

}

public bool IsUnique(string word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = new ValidWordAbbr(dictionary);
* bool param_1 = obj.IsUnique(word);
*/
```

C Solution:

```
/*
* Problem: Unique Word Abbreviation
* 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 {

} ValidWordAbbr;


ValidWordAbbr* validWordAbbrCreate(char** dictionary, int dictionarySize) {

}

bool validWordAbbrIsUnique(ValidWordAbbr* obj, char* word) {

}

void validWordAbbrFree(ValidWordAbbr* obj) {

}

/**
* Your ValidWordAbbr struct will be instantiated and called as such:
* ValidWordAbbr* obj = validWordAbbrCreate(dictionary, dictionarySize);
* bool param_1 = validWordAbbrIsUnique(obj, word);

* validWordAbbrFree(obj);
*/
```

Go Solution:

```
// Problem: Unique Word Abbreviation
// 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 ValidWordAbbr struct {

}


func Constructor(dictionary []string) ValidWordAbbr {

}


func (this *ValidWordAbbr) IsUnique(word string) bool {

}


/**
* Your ValidWordAbbr object will be instantiated and called as such:
* obj := Constructor(dictionary);
* param_1 := obj.IsUnique(word);
*/
```

Kotlin Solution:

```
class ValidWordAbbr(dictionary: Array<String>) {

fun isUnique(word: String): Boolean {

}

}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* var obj = ValidWordAbbr(dictionary)
* var param_1 = obj.isUnique(word)
*/
```

Swift Solution:

```
class ValidWordAbbr {

init(_ dictionary: [String]) {

}

func isUnique(_ word: String) -> Bool {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* let obj = ValidWordAbbr(dictionary)
* let ret_1: Bool = obj.isUnique(word)
*/
```

Rust Solution:

```
// Problem: Unique Word Abbreviation
// 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 ValidWordAbbr {

}


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

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

}
```

```
fn is_unique(&self, word: String) -> bool {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* let obj = ValidWordAbbr::new(dictionary);
* let ret_1: bool = obj.is_unique(word);
*/
```

Ruby Solution:

```
class ValidWordAbbr

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

end


=begin
:type word: String
:rtype: Boolean
=end
def is_unique(word)

end


end

# Your ValidWordAbbr object will be instantiated and called as such:
# obj = ValidWordAbbr.new(dictionary)
# param_1 = obj.is_unique(word)
```

PHP Solution:

```
class ValidWordAbbr {
/**
* @param String[] $dictionary
*/
function __construct($dictionary) {

}

/**
* @param String $word
* @return Boolean
*/
function isUnique($word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* $obj = ValidWordAbbr($dictionary);
* $ret_1 = $obj->isUnique($word);
*/
```

Dart Solution:

```
class ValidWordAbbr {

ValidWordAbbr(List<String> dictionary) {

}

bool isUnique(String word) {

}
}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* ValidWordAbbr obj = ValidWordAbbr(dictionary);
* bool param1 = obj.isUnique(word);
*/
```

Scala Solution:

```
class ValidWordAbbr(_dictionary: Array[String]) {

def isUnique(word: String): Boolean = {

}

}

/**
* Your ValidWordAbbr object will be instantiated and called as such:
* val obj = new ValidWordAbbr(dictionary)
* val param_1 = obj.isUnique(word)
*/
```

Elixir Solution:

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

end

@spec is_unique(word :: String.t) :: boolean
def is_unique(word) do

end
end

# Your functions will be called as such:
# ValidWordAbbr.init_(dictionary)
# param_1 = ValidWordAbbr.is_unique(word)

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

Erlang Solution:

```
-spec valid_word_abbr_init_(Dictionary :: [unicode:unicode_binary()]) ->
any().
valid_word_abbr_init_(Dictionary) ->
.
```

```
-spec valid_word_abbr_is_unique(Word :: unicode:unicode_binary()) ->
boolean().
valid_word_abbr_is_unique(Word) ->
.


%% Your functions will be called as such:
%% valid_word_abbr_init_(Dictionary),
%% Param_1 = valid_word_abbr_is_unique(Word),

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

Racket Solution:

```
(define valid-word-abbr%
(class object%
(super-new)

; dictionary : (listof string?)
(init-field
dictionary)

; is-unique : string? -> boolean?
(define/public (is-unique word)
)))

;; Your valid-word-abbr% object will be instantiated and called as such:
;; (define obj (new valid-word-abbr% [dictionary dictionary]))
;; (define param_1 (send obj is-unique word))
```