Problem 408: Valid Word Abbreviation

Problem Information

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

Problem Description

A string can be

abbreviated

by replacing any number of

non-adjacent

,

non-empty

substrings with their lengths. The lengths

should not

have leading zeros.

For example, a string such as

"substitution"

could be abbreviated as (but not limited to):

"s10n"

(

"s

ubstitutio

n"

)

"sub4u4"

(

"sub

stit

u

tion

"

)

"12"

(

"

substitution

"

)

"su3i1u2on"

(

"su

bst

i

t

u

ti

on"

)

"substitution"

(no substrings replaced)

The following are

not valid

abbreviations:

"s55n"

(

"s

ubsti

tutio

n"

, the replaced substrings are adjacent)

"s010n"

(has leading zeros)

"s0ubstitution"

(replaces an empty substring)

Given a string

word

and an abbreviation

abbr

, return

whether the string

matches

the given abbreviation

.

A

substring

is a contiguous

non-empty

sequence of characters within a string.

Example 1:

Input:

word = "internationalization", abbr = "i12iz4n"

Output:

true

Explanation:

The word "internationalization" can be abbreviated as "i12iz4n" ("i

nternational

iz

atio

n").

Example 2:

Input:

word = "apple", abbr = "a2e"

Output:

false

Explanation:

The word "apple" cannot be abbreviated as "a2e".

Constraints:

1 <= word.length <= 20

word

consists of only lowercase English letters.

1 <= abbr.length <= 10

abbr

consists of lowercase English letters and digits.

All the integers in

abbr

will fit in a 32-bit integer.

Code Snippets

C++:

```
class Solution {
public:
bool validWordAbbreviation(string word, string abbr) {

}
};
```

Java:

```
class Solution {
public boolean validWordAbbreviation(String word, String abbr) {

}
}
```

Python3:

```
class Solution:
def validWordAbbreviation(self, word: str, abbr: str) -> bool:
```

Python:

```
class Solution(object):
def validWordAbbreviation(self, word, abbr):
"""
:type word: str
:type abbr: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} word
* @param {string} abbr
* @return {boolean}
*/
var validWordAbbreviation = function(word, abbr) {

};
```

TypeScript:

```
function validWordAbbreviation(word: string, abbr: string): boolean {

};
```

C#:

```
public class Solution {
public bool ValidWordAbbreviation(string word, string abbr) {

}
}
```

C:

```
bool validWordAbbreviation(char* word, char* abbr) {

}
```

Go:

```
func validWordAbbreviation(word string, abbr string) bool {

}
```

Kotlin:

```
class Solution {
fun validWordAbbreviation(word: String, abbr: String): Boolean {

}
}
```

Swift:

```
class Solution {
func validWordAbbreviation(_ word: String, _ abbr: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn valid_word_abbreviation(word: String, abbr: String) -> bool {

}
}
```

Ruby:

```
# @param {String} word
# @param {String} abbr
# @return {Boolean}
def valid_word_abbreviation(word, abbr)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String $word
* @param String $abbr
* @return Boolean
*/
function validWordAbbreviation($word, $abbr) {

}
}
```

Dart:

```
class Solution {
bool validWordAbbreviation(String word, String abbr) {

}
}
```

Scala:

```
object Solution {
def validWordAbbreviation(word: String, abbr: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_word_abbreviation(word :: String.t, abbr :: String.t) :: boolean
def valid_word_abbreviation(word, abbr) do

end
end
```

Erlang:

```
-spec valid_word_abbreviation(Word :: unicode:unicode_binary(), Abbr ::
unicode:unicode_binary()) -> boolean().
valid_word_abbreviation(Word, Abbr) ->
.
```

Racket:

```
(define/contract (valid-word-abbreviation word abbr)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
bool validWordAbbreviation(string word, string abbr) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public boolean validWordAbbreviation(String word, String abbr) {
}
}
```

Python3 Solution:

```
"""
Problem: Valid Word Abbreviation
Difficulty: Easy
Tags: array, string, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def validWordAbbreviation(self, word: str, abbr: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def validWordAbbreviation(self, word, abbr):
"""
:type word: str
:type abbr: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* @param {string} word
* @param {string} abbr
* @return {boolean}
*/
var validWordAbbreviation = function(word, abbr) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function validWordAbbreviation(word: string, abbr: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public bool ValidWordAbbreviation(string word, string abbr) {
}
}
```

C Solution:

```
/*
* Problem: Valid Word Abbreviation
* Difficulty: Easy
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

bool validWordAbbreviation(char* word, char* abbr) {

}
```

Go Solution:

```
// Problem: Valid Word Abbreviation
// Difficulty: Easy
// Tags: array, string, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func validWordAbbreviation(word string, abbr string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validWordAbbreviation(word: String, abbr: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func validWordAbbreviation(_ word: String, _ abbr: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Word Abbreviation
// Difficulty: Easy
// Tags: array, string, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn valid_word_abbreviation(word: String, abbr: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} word
# @param {String} abbr
# @return {Boolean}
def valid_word_abbreviation(word, abbr)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @param String $abbr
* @return Boolean
*/
function validWordAbbreviation($word, $abbr) {

}
}
```

Dart Solution:

```
class Solution {
bool validWordAbbreviation(String word, String abbr) {

}
}
```

Scala Solution:

```
object Solution {
def validWordAbbreviation(word: String, abbr: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_word_abbreviation(word :: String.t, abbr :: String.t) :: boolean
def valid_word_abbreviation(word, abbr) do

end
end
```

Erlang Solution:

```
-spec valid_word_abbreviation(Word :: unicode:unicode_binary(), Abbr ::
unicode:unicode_binary()) -> boolean().
valid_word_abbreviation(Word, Abbr) ->
.
```

Racket Solution:

```
(define/contract (valid-word-abbreviation word abbr)
(-> string? string? boolean?)
)
```