Problem 291: Word Pattern II

Problem Information

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

Problem Description

Given a

pattern

and a string

s

, return

true

if

s

matches

the

pattern

.

A string

s

matches

a

pattern

if there is some

bijective mapping

of single characters to

non-empty

strings such that if each character in

pattern

is replaced by the string it maps to, then the resulting string is

s

. A

bijective mapping

means that no two characters map to the same string, and no character maps to two different strings.

Example 1:

Input:

pattern = "abab", s = "redblueredblue"

Output:

true

Explanation:

One possible mapping is as follows: 'a' -> "red" 'b' -> "blue"

Example 2:

Input:

pattern = "aaaa", s = "asdasdasdasd"

Output:

true

Explanation:

One possible mapping is as follows: 'a' -> "asd"

Example 3:

Input:

pattern = "aabb", s = "xyzabcxzyabc"

Output:

false

Constraints:

1 <= pattern.length, s.length <= 20

pattern

and

s

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool wordPatternMatch(string pattern, string s) {

}
};
```

Java:

```
class Solution {
public boolean wordPatternMatch(String pattern, String s) {

}
}
```

Python3:

```
class Solution:
def wordPatternMatch(self, pattern: str, s: str) -> bool:
```

Python:

```
class Solution(object):
def wordPatternMatch(self, pattern, s):
"""
:type pattern: str
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} pattern
* @param {string} s
* @return {boolean}
*/
var wordPatternMatch = function(pattern, s) {

};
```

TypeScript:

```
function wordPatternMatch(pattern: string, s: string): boolean {

};
```

C#:

```
public class Solution {
public bool WordPatternMatch(string pattern, string s) {

}
}
```

C:

```
bool wordPatternMatch(char* pattern, char* s) {

}
```

Go:

```
func wordPatternMatch(pattern string, s string) bool {

}
```

Kotlin:

```
class Solution {
fun wordPatternMatch(pattern: String, s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func wordPatternMatch(_ pattern: String, _ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn word_pattern_match(pattern: String, s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} pattern
# @param {String} s
# @return {Boolean}
def word_pattern_match(pattern, s)

end
```

PHP:

```
class Solution {

/**
* @param String $pattern
* @param String $s
* @return Boolean
*/
function wordPatternMatch($pattern, $s) {

}
}
```

Dart:

```
class Solution {
bool wordPatternMatch(String pattern, String s) {

}
}
```

Scala:

```
object Solution {
def wordPatternMatch(pattern: String, s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec word_pattern_match(pattern :: String.t, s :: String.t) :: boolean
def word_pattern_match(pattern, s) do

end
end
```

Erlang:

```
-spec word_pattern_match(Pattern :: unicode:unicode_binary(), S ::
unicode:unicode_binary()) -> boolean().
word_pattern_match(Pattern, S) ->
.
```

Racket:

```
(define/contract (word-pattern-match pattern s)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool wordPatternMatch(string pattern, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean wordPatternMatch(String pattern, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Pattern II
Difficulty: Medium
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
```

```
"""

class Solution:
def wordPatternMatch(self, pattern: str, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def wordPatternMatch(self, pattern, s):
"""
:type pattern: str
:type s: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} pattern
* @param {string} s
* @return {boolean}
*/
var wordPatternMatch = function(pattern, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function wordPatternMatch(pattern: string, s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool WordPatternMatch(string pattern, string s) {

}
}
```

C Solution:

```
/*
* Problem: Word Pattern II
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

bool wordPatternMatch(char* pattern, char* s) {

}
```

Go Solution:

```
// Problem: Word Pattern II
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func wordPatternMatch(pattern string, s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun wordPatternMatch(pattern: String, s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func wordPatternMatch(_ pattern: String, _ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Word Pattern II
// Difficulty: Medium
// Tags: string, hash
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn word_pattern_match(pattern: String, s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} pattern
# @param {String} s
# @return {Boolean}
def word_pattern_match(pattern, s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $pattern
* @param String $s
* @return Boolean
*/
function wordPatternMatch($pattern, $s) {

}
}
```

Dart Solution:

```
class Solution {
bool wordPatternMatch(String pattern, String s) {

}
}
```

Scala Solution:

```
object Solution {
def wordPatternMatch(pattern: String, s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec word_pattern_match(pattern :: String.t, s :: String.t) :: boolean
def word_pattern_match(pattern, s) do

end
end
```

Erlang Solution:

```
-spec word_pattern_match(Pattern :: unicode:unicode_binary(), S ::
unicode:unicode_binary()) -> boolean().
word_pattern_match(Pattern, S) ->
.
```

Racket Solution:

```
(define/contract (word-pattern-match pattern s)
(-> string? string? boolean?)
)
```