

Problem 290: Word Pattern

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a

pattern

and a string

s

, find if

s

follows the same pattern.

Here

follow

means a full match, such that there is a bijection between a letter in

pattern

and a

non-empty

word in

s

. Specifically:

Each letter in

pattern

maps to

exactly

one unique word in

s

.

Each unique word in

s

maps to

exactly

one letter in

pattern

.

No two letters map to the same word, and no two words map to the same letter.

Example 1:

Input:

```
pattern = "abba", s = "dog cat cat dog"
```

Output:

true

Explanation:

The bijection can be established as:

'a'

maps to

"dog"

.

'b'

maps to

"cat"

.

Example 2:

Input:

```
pattern = "abba", s = "dog cat cat fish"
```

Output:

false

Example 3:

Input:

pattern = "aaaa", s = "dog cat cat dog"

Output:

false

Constraints:

$1 \leq \text{pattern.length} \leq 300$

pattern

contains only lower-case English letters.

$1 \leq \text{s.length} \leq 3000$

s

contains only lowercase English letters and spaces

' '

.

s

does not contain

any leading or trailing spaces.

All the words in

s

are separated by a

single space

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
 * @param {string} pattern
 * @param {string} s
```

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

};
```

TypeScript:

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

};
```

C#:

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

    }
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

    }
}
```

Swift:

```

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

  }
}

```

Rust:

```

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

  }
}

```

Ruby:

```

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

end

```

PHP:

```

class Solution {

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

  }
}

```

Dart:

```

class Solution {
  bool wordPattern(String pattern, String s) {

  }
}

```

```
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*  
 * Problem: Word Pattern  
 * Difficulty: Easy  
 * 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 wordPattern(string pattern, string s) {

    }
};

```

Java Solution:

```

/**
 * Problem: Word Pattern
 * Difficulty: Easy
 * 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 wordPattern(String pattern, String s) {

    }
}

```

Python3 Solution:

```

"""
Problem: Word Pattern
Difficulty: Easy
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 wordPattern(self, pattern: str, s: str) -> bool:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Word Pattern
 * Difficulty: Easy
 * 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 wordPattern = function(pattern, s) {

};

```

TypeScript Solution:

```

/**
 * Problem: Word Pattern
 * Difficulty: Easy
 * 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 wordPattern(pattern: string, s: string): boolean {

};

```

C# Solution:

```

/*
 * Problem: Word Pattern
 * Difficulty: Easy
 * 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 WordPattern(string pattern, string s) {

    }
}

```

C Solution:

```

/*
 * Problem: Word Pattern
 * Difficulty: Easy
 * 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 wordPattern(char* pattern, char* s) {

}

```

Go Solution:

```

// Problem: Word Pattern
// Difficulty: Easy
// 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 wordPattern(pattern string, s string) bool {

}

```

Kotlin Solution:

```

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

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```

// Problem: Word Pattern
// Difficulty: Easy
// 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(pattern: String, s: String) -> bool {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

    }
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

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