'

Problem 3407: Substring Matching Pattern

Problem Information

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

Problem Description

You are given a string

s

and a pattern string

p

, where

p

contains

exactly one

'*'

character.

The

'*'

in

p

can be replaced with any sequence of zero or more characters.

Return

true

if

p

can be made a

substring

of

s

, and

false

otherwise.

Example 1:

Input:

s = "leetcode", p = "ee*e"

Output:

true

Explanation:

By replacing the

'*'

with

"tcod"

, the substring

"eetcode"

matches the pattern.

Example 2:

Input:

s = "car", p = "c*v"

Output:

false

Explanation:

There is no substring matching the pattern.

Example 3:

Input:

s = "luck", p = "u*"

Output:

true

Explanation:

The substrings

"u"

,

"uc"

, and

"uck"

match the pattern.

Constraints:

1 <= s.length <= 50

1 <= p.length <= 50

s

contains only lowercase English letters.

p

contains only lowercase English letters and exactly one

'*'

Code Snippets

C++:

```
class Solution {
public:
bool hasMatch(string s, string p) {
}
};
```

Java:

```
class Solution {
public boolean hasMatch(String s, String p) {

}
}
```

Python3:

```
class Solution:
def hasMatch(self, s: str, p: str) -> bool:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function hasMatch(s: string, p: string): boolean {

};
```

C#:

```
public class Solution {
public bool HasMatch(string s, string p) {

}
}
```

C:

```
bool hasMatch(char* s, char* p) {

}
```

Go:

```
func hasMatch(s string, p string) bool {

}
```

Kotlin:

```
class Solution {
fun hasMatch(s: String, p: String): Boolean {

}
}
```

Swift:

```
class Solution {
func hasMatch(_ s: String, _ p: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn has_match(s: String, p: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} p
# @return {Boolean}
def has_match(s, p)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $p
* @return Boolean
*/
function hasMatch($s, $p) {

}
}
```

Dart:

```
class Solution {
bool hasMatch(String s, String p) {

}
}
```

Scala:

```
object Solution {
def hasMatch(s: String, p: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec has_match(s :: String.t, p :: String.t) :: boolean
```

```
def has_match(s, p) do

end
end
```

Erlang:

```
-spec has_match(S :: unicode:unicode_binary(), P :: unicode:unicode_binary())
-> boolean().
has_match(S, P) ->
.
```

Racket:

```
(define/contract (has-match s p)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
bool hasMatch(string s, string p) {

}
};
```

Java Solution:

```
/**
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public boolean hasMatch(String s, String p) {

}
}
```

Python3 Solution:

```
"""
Problem: Substring Matching Pattern
Difficulty: Easy
Tags: string, tree

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function hasMatch(s: string, p: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* 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 HasMatch(string s, string p) {

}
}
```

C Solution:

```
/*
* Problem: Substring Matching Pattern
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

bool hasMatch(char* s, char* p) {

}
```

Go Solution:

```
// Problem: Substring Matching Pattern
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func hasMatch(s string, p string) bool {
}
```

Kotlin Solution:

```
class Solution {
fun hasMatch(s: String, p: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func hasMatch(_ s: String, _ p: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Substring Matching Pattern
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn has_match(s: String, p: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} p
# @return {Boolean}
def has_match(s, p)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $p
* @return Boolean
*/
function hasMatch($s, $p) {

}
}
```

Dart Solution:

```
class Solution {
bool hasMatch(String s, String p) {

}
}
```

Scala Solution:

```
object Solution {
def hasMatch(s: String, p: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec has_match(S :: unicode:unicode_binary(), P :: unicode:unicode_binary())
-> boolean().
has_match(S, P) ->
.
```

Racket Solution:

```
(define/contract (has-match s p)
(-> string? string? boolean?)
)
```