Problem 10: Regular Expression Matching

Problem Information

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

Problem Description

Given an input string

s

and a pattern

p

, implement regular expression matching with support for

'.'

and

'*'

where:

'.'

Matches any single character.

'*'

Matches zero or more of the preceding element.

The matching should cover the

entire

input string (not partial).

Example 1:

Input:

s = "aa", p = "a"

Output:

false

Explanation:

"a" does not match the entire string "aa".

Example 2:

Input:

s = "aa", p = "a*"

Output:

true

Explanation:

'*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:

Input:

s = "ab", p = ".*"

Output:

true

Explanation:

".*" means "zero or more (*) of any character (.)".

Constraints:

1 <= s.length <= 20

1 <= p.length <= 20

s

contains only lowercase English letters.

p

contains only lowercase English letters,

'.'

, and

'*'

.

It is guaranteed for each appearance of the character

'*'

, there will be a previous valid character to match.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Regular Expression Matching
Difficulty: Hard
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool IsMatch(string s, string p) {

}
}
```

C Solution:

```
/*
* Problem: Regular Expression Matching
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Regular Expression Matching
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func isMatch(s string, p string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Regular Expression Matching
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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