Problem 3455: Shortest Matching Substring

Problem Information

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

Problem Description

You are given a string

s

and a pattern string

p

, where

p

contains

exactly two

'*'

characters.

The

'*'

in

p

matches any sequence of zero or more characters.

Return the length of the

shortest

substring

in

s

that matches

p

. If there is no such substring, return -1.

Note:

The empty substring is considered valid.

Example 1:

Input:

s = "abaacbaecebce", p = "ba*c*ce"

Output:

8

Explanation:

The shortest matching substring of

p

in

s

is

"

ba

e

c

eb

ce

"

.

Example 2:

Input:

s = "baccbaadbc", p = "cc*baa*adb"

Output:

-1

Explanation:

There is no matching substring in

s

.

Example 3:

Input:

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

Output:

0

Explanation:

The empty substring is the shortest matching substring.

Example 4:

Input:

s = "madlogic", p = "*adlogi*"

Output:

6

Explanation:

The shortest matching substring of

p

in

s

is

"

adlogi

"

.

Constraints:

1 <= s.length <= 10

5

2 <= p.length <= 10

5

s

contains only lowercase English letters.

p

contains only lowercase English letters and exactly two

'*'

.

Code Snippets

C++:

```
class Solution {
public:
int shortestMatchingSubstring(string s, string p) {

}
};
```

Java:

```
class Solution {
public int shortestMatchingSubstring(String s, String p) {

}
}
```

Python3:

```
class Solution:
def shortestMatchingSubstring(self, s: str, p: str) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestMatchingSubstring(s: string, p: string): number {

};
```

C#:

```
public class Solution {
public int ShortestMatchingSubstring(string s, string p) {

}
}
```

C:

```
int shortestMatchingSubstring(char* s, char* p) {

}
```

Go:

```
func shortestMatchingSubstring(s string, p string) int {

}
```

Kotlin:

```
class Solution {
fun shortestMatchingSubstring(s: String, p: String): Int {

}
}
```

Swift:

```
class Solution {
func shortestMatchingSubstring(_ s: String, _ p: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_matching_substring(s: String, p: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} p
# @return {Integer}
def shortest_matching_substring(s, p)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $p
* @return Integer
*/
function shortestMatchingSubstring($s, $p) {

}
}
```

Dart:

```
class Solution {
int shortestMatchingSubstring(String s, String p) {

}
}
```

Scala:

```
object Solution {
def shortestMatchingSubstring(s: String, p: String): Int = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec shortest_matching_substring(S :: unicode:unicode_binary(), P ::
unicode:unicode_binary()) -> integer().
shortest_matching_substring(S, P) ->
.
```

Racket:

```
(define/contract (shortest-matching-substring s p)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Matching Substring
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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:
int shortestMatchingSubstring(string s, string p) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Matching Substring
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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 int shortestMatchingSubstring(String s, String p) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Matching Substring
Difficulty: Hard
Tags: array, string, tree, search

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 shortestMatchingSubstring(self, s: str, p: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Matching Substring
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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} s
* @param {string} p
* @return {number}
*/
var shortestMatchingSubstring = function(s, p) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Matching Substring
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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 shortestMatchingSubstring(s: string, p: string): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Matching Substring
* Difficulty: Hard
```

```
* Tags: array, string, tree, search
*
* 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 int ShortestMatchingSubstring(string s, string p) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Matching Substring
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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
*/

int shortestMatchingSubstring(char* s, char* p) {

}
```

Go Solution:

```
// Problem: Shortest Matching Substring
// Difficulty: Hard
// Tags: array, string, tree, search
//
// 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 shortestMatchingSubstring(s string, p string) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun shortestMatchingSubstring(s: String, p: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func shortestMatchingSubstring(_ s: String, _ p: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Matching Substring
// Difficulty: Hard
// Tags: array, string, tree, search
//
// 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 shortest_matching_substring(s: String, p: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} p
# @return {Integer}
def shortest_matching_substring(s, p)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $p
* @return Integer
*/
function shortestMatchingSubstring($s, $p) {

}
}
```

Dart Solution:

```
class Solution {
int shortestMatchingSubstring(String s, String p) {

}
}
```

Scala Solution:

```
object Solution {
def shortestMatchingSubstring(s: String, p: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec shortest_matching_substring(S :: unicode:unicode_binary(), P ::
unicode:unicode_binary()) -> integer().
shortest_matching_substring(S, P) ->
.
```

Racket Solution:

```
(define/contract (shortest-matching-substring s p)
(-> string? string? exact-integer?)
)
```