Problem 2055: Plates Between Candles

Problem Information

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

Problem Description

There is a long table with a line of plates and candles arranged on top of it. You are given a

0-indexed

string

s

consisting of characters

'*'

and

'|'

only, where a

'*'

represents a

plate

and a

'|'

represents a

candle

.

You are also given a

0-indexed

2D integer array

queries

where

queries[i] = [left

i

, right

i

]

denotes the

substring

s[left

i

...right

i

]

(

inclusive

). For each query, you need to find the

number

of plates

between candles

that are

in the substring

. A plate is considered

between candles

if there is at least one candle to its left

and

at least one candle to its right

in the substring

.

For example,

s = "||**||**|*"

, and a query

[3, 8]

denotes the substring

"*||

**

|"

. The number of plates between candles in this substring is

2

, as each of the two plates has at least one candle

in the substring

to its left

and

right.

Return

an integer array

answer

where

answer[i]

is the answer to the

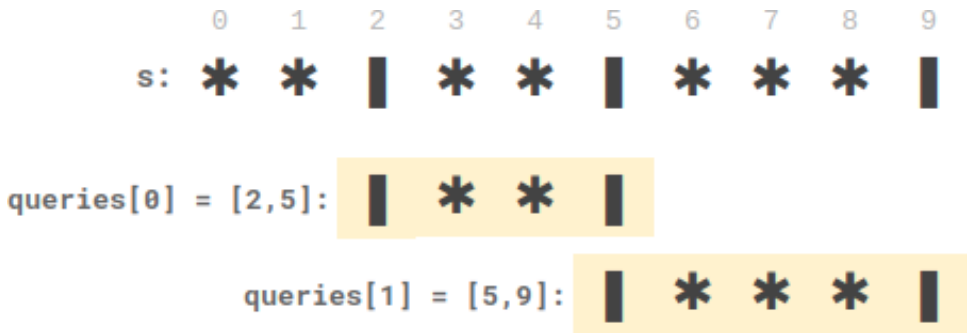
i

th

query

.

Example 1:



Input:

s = "**|**|***|", queries = [[2,5],[5,9]]

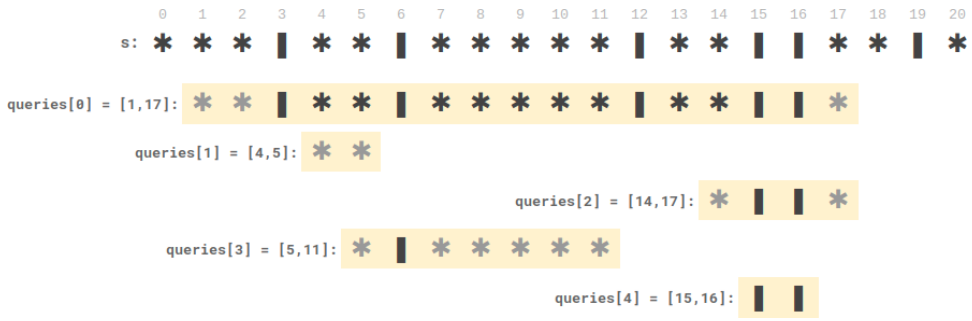
Output:

[2,3]

Explanation:

- queries[0] has two plates between candles. - queries[1] has three plates between candles.

Example 2:



Input:

s = "***|**|*****|**||**|*", queries = [[1,17],[4,5],[14,17],[5,11],[15,16]]

Output:

[9,0,0,0,0]

Explanation:

- queries[0] has nine plates between candles. - The other queries have zero plates between candles.

Constraints:

3 <= s.length <= 10

5

s

consists of

'*'

and

'|'

characters.

1 <= queries.length <= 10

5

queries[i].length == 2

0 <= left

i

<= right

i

< s.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> platesBetweenCandles(string s, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] platesBetweenCandles(String s, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def platesBetweenCandles(self, s: str, queries: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def platesBetweenCandles(self, s, queries):
"""
:type s: str
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[][]} queries
* @return {number[]}
*/
var platesBetweenCandles = function(s, queries) {

};
```

TypeScript:

```
function platesBetweenCandles(s: string, queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] PlatesBetweenCandles(string s, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* platesBetweenCandles(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go:

```
func platesBetweenCandles(s string, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun platesBetweenCandles(s: String, queries: Array<IntArray>): IntArray {
```

```
}
}
```

Swift:

```
class Solution {
func platesBetweenCandles(_ s: String, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn plates_between_candles(s: String, queries: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Integer[]}
def plates_between_candles(s, queries)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $queries
* @return Integer[]
*/
function platesBetweenCandles($s, $queries) {

}
```

```
}
```

Dart:

```
class Solution {
List<int> platesBetweenCandles(String s, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def platesBetweenCandles(s: String, queries: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec plates_between_candles(s :: String.t, queries :: [[integer]]) ::
[integer]
def plates_between_candles(s, queries) do

end
end
```

Erlang:

```
-spec plates_between_candles(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [integer()].
plates_between_candles(S, Queries) ->
.
```

Racket:

```
(define/contract (plates-between-candles s queries)
(-> string? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Plates Between Candles
* Difficulty: Medium
* 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:
vector<int> platesBetweenCandles(string s, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Plates Between Candles
* Difficulty: Medium
* 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[] platesBetweenCandles(String s, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Plates Between Candles
Difficulty: Medium
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 platesBetweenCandles(self, s: str, queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def platesBetweenCandles(self, s, queries):
"""
:type s: str
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Plates Between Candles
* Difficulty: Medium
* 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 {number[][]} queries
* @return {number[]}
```

```
*/
var platesBetweenCandles = function(s, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Plates Between Candles
* Difficulty: Medium
* 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 platesBetweenCandles(s: string, queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Plates Between Candles
* Difficulty: Medium
* 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[] PlatesBetweenCandles(string s, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Plates Between Candles
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* platesBetweenCandles(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Plates Between Candles
// Difficulty: Medium
// 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 platesBetweenCandles(s string, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun platesBetweenCandles(s: String, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func platesBetweenCandles(_ s: String, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Plates Between Candles
// Difficulty: Medium
// 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 plates_between_candles(s: String, queries: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Integer[]}
def plates_between_candles(s, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $queries
* @return Integer[]
*/
function platesBetweenCandles($s, $queries) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> platesBetweenCandles(String s, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def platesBetweenCandles(s: String, queries: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec plates_between_candles(s :: String.t, queries :: [[integer]]) ::
[integer]
def plates_between_candles(s, queries) do

end
end
```

Erlang Solution:

```
-spec plates_between_candles(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [integer()].
plates_between_candles(S, Queries) ->
.
```

Racket Solution:

```
(define/contract (plates-between-candles s queries)
(-> string? (listof (listof exact-integer?)) (listof exact-integer?))
)
```