Problem 3529: Count Cells in Overlapping Horizontal and Vertical Substrings

Problem Information

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

Problem Description

You are given an

m x n

matrix

grid

consisting of characters and a string

pattern

.

A

horizontal substring

is a contiguous sequence of characters read from left to right. If the end of a row is reached before the substring is complete, it wraps to the first column of the next row and continues as needed. You do

not

wrap from the bottom row back to the top.

A

vertical substring

is a contiguous sequence of characters read from top to bottom. If the bottom of a column is reached before the substring is complete, it wraps to the first row of the next column and continues as needed. You do

not

wrap from the last column back to the first.

Count the number of cells in the matrix that satisfy the following condition:

The cell must be part of

at least

one horizontal substring and

at least

one vertical substring, where

both

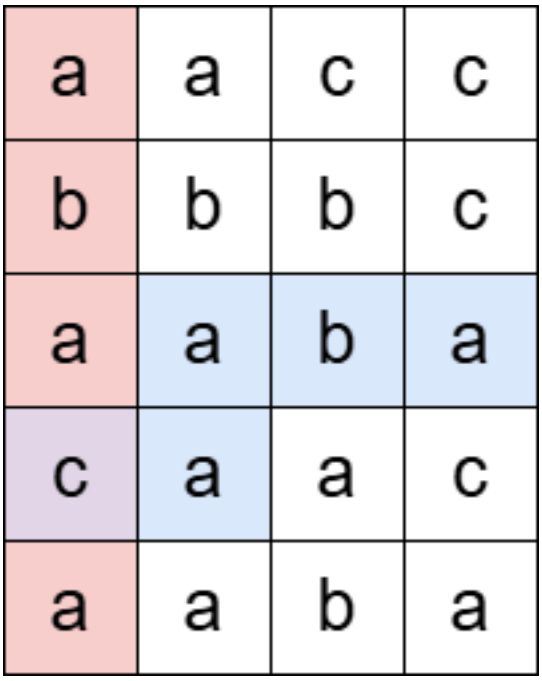
substrings are equal to the given

pattern

.

Return the count of these cells.

Example 1:



Input:

grid = [["a","a","c","c"],["b","b","b","c"],["a","a","b","a"],["c","a","a","c"],["a","a","b","a"]], pattern = "abaca"

Output:

1

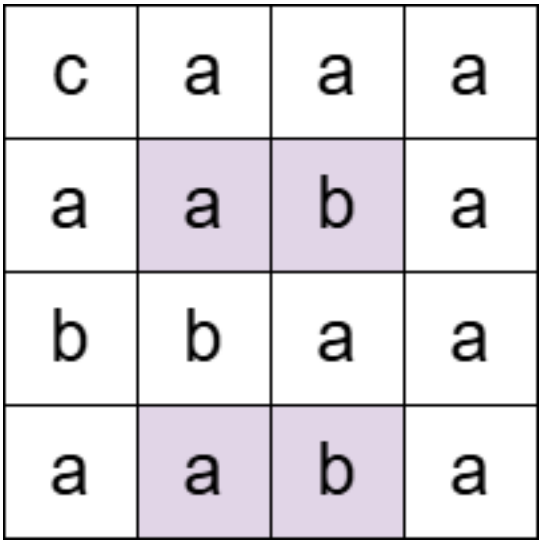
Explanation:

The pattern

"abaca"

appears once as a horizontal substring (colored blue) and once as a vertical substring (colored red), intersecting at one cell (colored purple).

Example 2:



Input:

grid = [["c","a","a","a"],["a","a","b","a"],["b","b","a","a"],["a","a","b","a"]], pattern = "aba"

Output:

4

Explanation:

The cells colored above are all part of at least one horizontal and one vertical substring matching the pattern

"aba"

.

Example 3:

Input:

grid = [["a"]], pattern = "a"

Output:

1

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m * n <= 10

5

1 <= pattern.length <= m * n

grid

and

pattern

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int countCells(vector<vector<char>>& grid, string pattern) {

}
};
```

Java:

```
class Solution {
public int countCells(char[][] grid, String pattern) {

}
```

```
}
```

Python3:

```
class Solution:
def countCells(self, grid: List[List[str]], pattern: str) -> int:
```

Python:

```
class Solution(object):
def countCells(self, grid, pattern):
"""
:type grid: List[List[str]]
:type pattern: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {character[][]} grid
* @param {string} pattern
* @return {number}
*/
var countCells = function(grid, pattern) {

};
```

TypeScript:

```
function countCells(grid: string[][], pattern: string): number {

};
```

C#:

```
public class Solution {
public int CountCells(char[][] grid, string pattern) {

}
}
```

C:

```
int countCells(char** grid, int gridSize, int* gridColSize, char* pattern) {

}
```

Go:

```
func countCells(grid [][]byte, pattern string) int {

}
```

Kotlin:

```
class Solution {
fun countCells(grid: Array<CharArray>, pattern: String): Int {

}
}
```

Swift:

```
class Solution {
func countCells(_ grid: [[Character]], _ pattern: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_cells(grid: Vec<Vec<char>>, pattern: String) -> i32 {

}
}
```

Ruby:

```
# @param {Character[][]} grid
# @param {String} pattern
# @return {Integer}
def count_cells(grid, pattern)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String[][] $grid
* @param String $pattern
* @return Integer
*/
function countCells($grid, $pattern) {

}
}
```

Dart:

```
class Solution {
int countCells(List<List<String>> grid, String pattern) {

}
}
```

Scala:

```
object Solution {
def countCells(grid: Array[Array[Char]], pattern: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_cells(grid :: [[char]], pattern :: String.t) :: integer
def count_cells(grid, pattern) do

end
end
```

Erlang:

```
-spec count_cells(Grid :: [[char()]], Pattern :: unicode:unicode_binary()) ->
integer().
count_cells(Grid, Pattern) ->
.
```

Racket:

```
(define/contract (count-cells grid pattern)
(-> (listof (listof char?)) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 countCells(vector<vector<char>>& grid, string pattern) {

}
};
```

Java Solution:

```
/**
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 countCells(char[][] grid, String pattern) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
Difficulty: Medium
Tags: array, string, tree, hash

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 countCells(self, grid: List[List[str]], pattern: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countCells(self, grid, pattern):
"""
:type grid: List[List[str]]
:type pattern: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
```

```
* Tags: array, string, tree, hash
*
* 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 {character[][]} grid
* @param {string} pattern
* @return {number}
*/
var countCells = function(grid, pattern) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 countCells(grid: string[][], pattern: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 CountCells(char[][] grid, string pattern) {

}
}
```

C Solution:

```
/*
* Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 countCells(char** grid, int gridSize, int* gridColSize, char* pattern) {

}
```

Go Solution:

```
// Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// 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 countCells(grid [][]byte, pattern string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCells(grid: Array<CharArray>, pattern: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countCells(_ grid: [[Character]], _ pattern: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Cells in Overlapping Horizontal and Vertical Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// 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 count_cells(grid: Vec<Vec<char>>, pattern: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} grid
# @param {String} pattern
# @return {Integer}
def count_cells(grid, pattern)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $grid
* @param String $pattern
* @return Integer
*/
function countCells($grid, $pattern) {

}
}
```

Dart Solution:

```
class Solution {
int countCells(List<List<String>> grid, String pattern) {

}
}
```

Scala Solution:

```
object Solution {
def countCells(grid: Array[Array[Char]], pattern: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_cells(grid :: [[char]], pattern :: String.t) :: integer
def count_cells(grid, pattern) do

end
end
```

Erlang Solution:

```
-spec count_cells(Grid :: [[char()]], Pattern :: unicode:unicode_binary()) ->
integer().
count_cells(Grid, Pattern) ->
```

```
.
```

Racket Solution:

```
(define/contract (count-cells grid pattern)
(-> (listof (listof char?)) string? exact-integer?)
)
```