Problem 3359: Find Sorted Submatrices With Maximum Element at Most K

Problem Information

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

Problem Description

You are given a 2D matrix

grid

of size

m x n

. You are also given a

non-negative

integer

k

.

Return the number of

submatrices

of

grid

that satisfy the following conditions:

The maximum element in the submatrix

less than or equal to

k

.

Each row in the submatrix is sorted in

non-increasing

order.

A submatrix

(x1, y1, x2, y2)

is a matrix that forms by choosing all cells

grid[x][y]

where

x1 <= x <= x2

and

y1 <= y <= y2

.

Example 1:

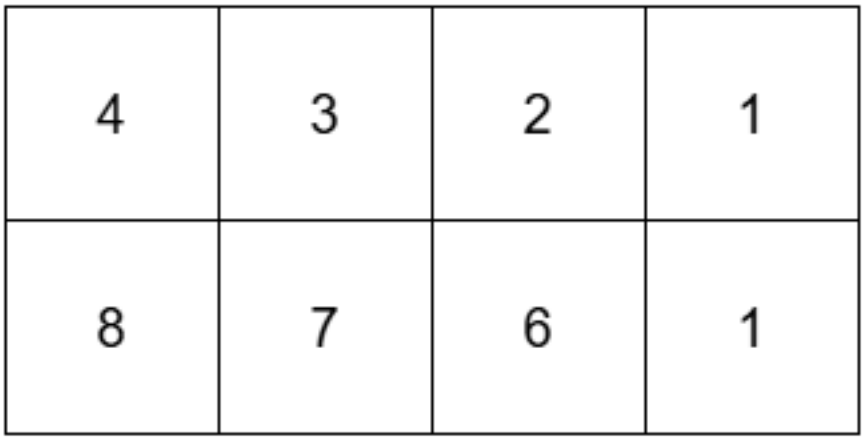
Input:

grid = [[4,3,2,1],[8,7,6,1]], k = 3

Output:

8

Explanation:



The 8 submatrices are:

[[1]]

[[1]]

[[2,1]]

[[3,2,1]]

[[1],[1]]

[[2]]

[[3]]

[[3,2]]

Example 2:

Input:

grid = [[1,1,1],[1,1,1],[1,1,1]], k = 1

Output:

36

Explanation:

There are 36 submatrices of grid. All submatrices have their maximum element equal to 1.

Example 3:

Input:

grid = [[1]], k = 1

Output:

1

Constraints:

1 <= m == grid.length <= 10

3

1 <= n == grid[i].length <= 10

3

1 <= grid[i][j] <= 10

9

1 <= k <= 10

Code Snippets

C++:

```
class Solution {
public:
long long countSubmatrices(vector<vector<int>>& grid, int k) {

}
};
```

Java:

```
class Solution {
public long countSubmatrices(int[][] grid, int k) {

}
}
```

Python3:

```
class Solution:
def countSubmatrices(self, grid: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def countSubmatrices(self, grid, k):
"""
:type grid: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
```

```
* @param {number} k
* @return {number}
*/
var countSubmatrices = function(grid, k) {

};
```

TypeScript:

```
function countSubmatrices(grid: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public long CountSubmatrices(int[][] grid, int k) {

}
}
```

C:

```
long long countSubmatrices(int** grid, int gridSize, int* gridColSize, int k)
{

}
```

Go:

```
func countSubmatrices(grid [][]int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun countSubmatrices(grid: Array<IntArray>, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func countSubmatrices(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_submatrices(grid: Vec<Vec<i32>>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def count_submatrices(grid, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer
*/
function countSubmatrices($grid, $k) {

}
}
```

Dart:

```
class Solution {
int countSubmatrices(List<List<int>> grid, int k) {
```

```
}
}
```

Scala:

```
object Solution {
def countSubmatrices(grid: Array[Array[Int]], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_submatrices(grid :: [[integer]], k :: integer) :: integer
def count_submatrices(grid, k) do

end
end
```

Erlang:

```
-spec count_submatrices(Grid :: [[integer()]], K :: integer()) -> integer().
count_submatrices(Grid, K) ->
.
```

Racket:

```
(define/contract (count-submatrices grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
```

```
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long countSubmatrices(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long countSubmatrices(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Sorted Submatrices With Maximum Element at Most K
Difficulty: Hard
Tags: array, sort, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countSubmatrices(self, grid: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countSubmatrices(self, grid, k):
"""
:type grid: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} grid
* @param {number} k
* @return {number}
*/
var countSubmatrices = function(grid, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countSubmatrices(grid: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long CountSubmatrices(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find Sorted Submatrices With Maximum Element at Most K
* Difficulty: Hard
* Tags: array, sort, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

long long countSubmatrices(int** grid, int gridSize, int* gridColSize, int k)
{

}
```

Go Solution:

```
// Problem: Find Sorted Submatrices With Maximum Element at Most K
// Difficulty: Hard
// Tags: array, sort, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countSubmatrices(grid [][]int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countSubmatrices(grid: Array<IntArray>, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func countSubmatrices(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Sorted Submatrices With Maximum Element at Most K
// Difficulty: Hard
```

```
// Tags: array, sort, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_submatrices(grid: Vec<Vec<i32>>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def count_submatrices(grid, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer
*/
function countSubmatrices($grid, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countSubmatrices(List<List<int>> grid, int k) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def countSubmatrices(grid: Array[Array[Int]], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_submatrices(grid :: [[integer]], k :: integer) :: integer
def count_submatrices(grid, k) do

end
end
```

Erlang Solution:

```
-spec count_submatrices(Grid :: [[integer()]], K :: integer()) -> integer().
count_submatrices(Grid, K) ->
.
```

Racket Solution:

```
(define/contract (count-submatrices grid k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```