Problem 3070: Count Submatrices with Top-Left Element and Sum Less Than k

Problem Information

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

Problem Description

You are given a

0-indexed

integer matrix

grid

and an integer

k

.

Return

the

number

of

submatrices

that contain the top-left element of the grid

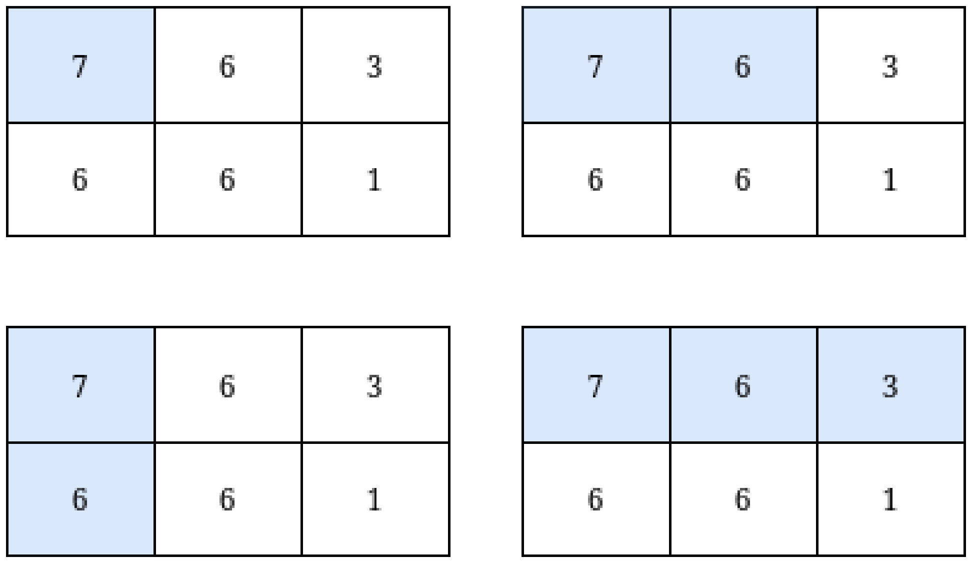
,

and have a sum less than or equal to

k

.

Example 1:



Input:

grid = [[7,6,3],[6,6,1]], k = 18

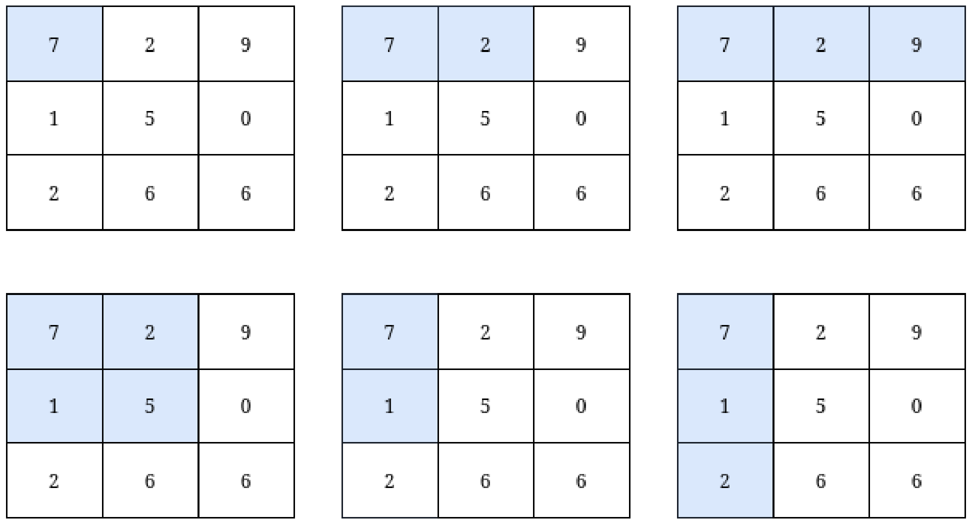
Output:

4

Explanation:

There are only 4 submatrices, shown in the image above, that contain the top-left element of grid, and have a sum less than or equal to 18.

Example 2:



Input:

grid = [[7,2,9],[1,5,0],[2,6,6]], k = 20

Output:

6

Explanation:

There are only 6 submatrices, shown in the image above, that contain the top-left element of grid, and have a sum less than or equal to 20.

Constraints:

m == grid.length

n == grid[i].length

1 <= n, m <= 1000

0 <= grid[i][j] <= 1000

1 <= k <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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


}
}
```

Rust:

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

}
}
```

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): Int = {

}
}
```

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: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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:
int countSubmatrices(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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 int countSubmatrices(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Submatrices with Top-Left Element and Sum Less Than k
Difficulty: Medium
Tags: array

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: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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 int CountSubmatrices(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Submatrices with Top-Left Element and Sum Less Than k
* Difficulty: Medium
* Tags: array
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Count Submatrices with Top-Left Element and Sum Less Than k
// Difficulty: Medium
// Tags: array
//
// 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) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Submatrices with Top-Left Element and Sum Less Than k
// Difficulty: Medium
// Tags: array
//
// 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) -> i32 {



}
}
```

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): Int = {

}
}
```

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?)
)
```