Problem 1504: Count Submatrices With All Ones

Problem Information

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

Problem Description

Given an

m x n

binary matrix

mat

,

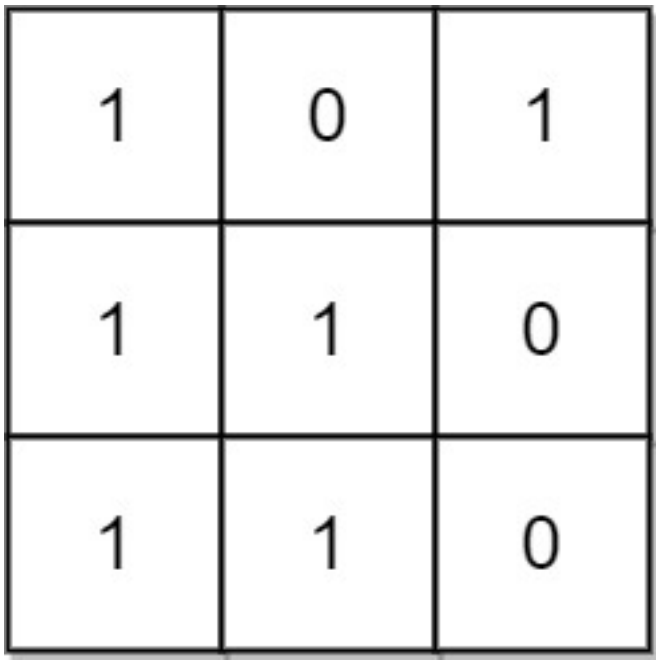
return the number of

submatrices

that have all ones

.

Example 1:



Input:

mat = [[1,0,1],[1,1,0],[1,1,0]]

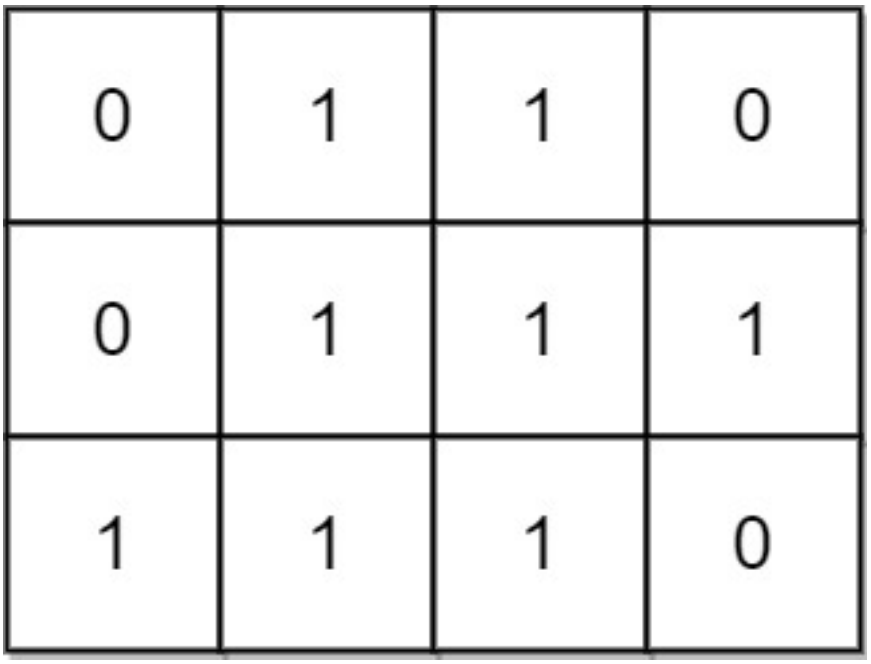
Output:

13

Explanation:

There are 6 rectangles of side 1x1. There are 2 rectangles of side 1x2. There are 3 rectangles of side 2x1. There is 1 rectangle of side 2x2. There is 1 rectangle of side 3x1. Total number of rectangles = 6 + 2 + 3 + 1 + 1 = 13.

Example 2:



Input:

mat = [[0,1,1,0],[0,1,1,1],[1,1,1,0]]

Output:

24

Explanation:

There are 8 rectangles of side 1x1. There are 5 rectangles of side 1x2. There are 2 rectangles of side 1x3. There are 4 rectangles of side 2x1. There are 2 rectangles of side 2x2. There are 2 rectangles of side 3x1. There is 1 rectangle of side 3x2. Total number of rectangles = 8 + 5 + 2 + 4 + 2 + 2 + 1 = 24.

Constraints:

1 <= m, n <= 150

mat[i][j]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int numSubmat(vector<vector<int>>& mat) {

}
};
```

Java:

```
class Solution {
public int numSubmat(int[][] mat) {

}
}
```

Python3:

```
class Solution:
def numSubmat(self, mat: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def numSubmat(self, mat):
"""
:type mat: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} mat
* @return {number}
*/
var numSubmat = function(mat) {

};
```

TypeScript:

```
function numSubmat(mat: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumSubmat(int[][] mat) {

}
}
```

C:

```
int numSubmat(int** mat, int matSize, int* matColSize) {

}
```

Go:

```
func numSubmat(mat [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numSubmat(mat: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func numSubmat(_ mat: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_submat(mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @return {Integer}
def num_submat(mat)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer
*/
function numSubmat($mat) {

}
}
```

Dart:

```
class Solution {
int numSubmat(List<List<int>> mat) {

}
}
```

Scala:

```
object Solution {
def numSubmat(mat: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_submat(mat :: [[integer]]) :: integer
def num_submat(mat) do

end
end
```

Erlang:

```
-spec num_submat(Mat :: [[integer()]]) -> integer().
num_submat(Mat) ->
.
```

Racket:

```
(define/contract (num-submat mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int numSubmat(vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numSubmat(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Submatrices With All Ones
Difficulty: Medium
Tags: array, dp, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numSubmat(self, mat: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def numSubmat(self, mat):
"""
:type mat: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} mat
* @return {number}
*/
var numSubmat = function(mat) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

function numSubmat(mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumSubmat(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: Count Submatrices With All Ones
* Difficulty: Medium
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numSubmat(int** mat, int matSize, int* matColSize) {

}
```

Go Solution:

```
// Problem: Count Submatrices With All Ones
// Difficulty: Medium
// Tags: array, dp, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numSubmat(mat [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSubmat(mat: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSubmat(_ mat: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Submatrices With All Ones
// Difficulty: Medium
// Tags: array, dp, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn num_submat(mat: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @return {Integer}
def num_submat(mat)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer
*/
function numSubmat($mat) {

}
}
```

Dart Solution:

```
class Solution {
int numSubmat(List<List<int>> mat) {

}
}
```

Scala Solution:

```
object Solution {
def numSubmat(mat: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_submat(mat :: [[integer]]) :: integer
def num_submat(mat) do

end
end
```

Erlang Solution:

```
-spec num_submat(Mat :: [[integer()]]) -> integer().
num_submat(Mat) ->
.
```

Racket Solution:

```
(define/contract (num-submat mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```