

Problem 1277: Count Square Submatrices with All Ones

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a

$m \times n$

matrix of ones and zeros, return how many

square

submatrices have all ones.

Example 1:

Input:

matrix = [[0,1,1,1], [1,1,1,1], [0,1,1,1]]

Output:

15

Explanation:

There are

10

squares of side 1. There are

4

squares of side 2. There is

1

square of side 3. Total number of squares = $10 + 4 + 1 =$

15

.

Example 2:

Input:

matrix = [[1,0,1], [1,1,0], [1,1,0]]

Output:

7

Explanation:

There are

6

squares of side 1. There is

1

square of side 2. Total number of squares = $6 + 1 =$

7

.

Constraints:

$1 \leq \text{arr.length} \leq 300$

$1 \leq \text{arr}[0].\text{length} \leq 300$

$0 \leq \text{arr}[i][j] \leq 1$

Code Snippets

C++:

```
class Solution {
public:
    int countSquares(vector<vector<int>>& matrix) {

    }
};
```

Java:

```
class Solution {
    public int countSquares(int[][] matrix) {

    }
}
```

Python3:

```
class Solution:
    def countSquares(self, matrix: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
    def countSquares(self, matrix):
        """
        :type matrix: List[List[int]]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countSquares(matrix: number[][]): number {

};
```

C#:

```
public class Solution {
    public int CountSquares(int[][] matrix) {

    }
}
```

C:

```
int countSquares(int** matrix, int matrixSize, int* matrixColSize) {

}
```

Go:

```
func countSquares(matrix [][]int) int {

}
```

Kotlin:

```

class Solution {
    fun countSquares(matrix: Array<IntArray>): Int {

    }
}

```

Swift:

```

class Solution {
    func countSquares(_ matrix: [[Int]]) -> Int {

    }
}

```

Rust:

```

impl Solution {
    pub fn count_squares(matrix: Vec<Vec<i32>>) -> i32 {

    }
}

```

Ruby:

```

# @param {Integer[][]} matrix
# @return {Integer}
def count_squares(matrix)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[][] $matrix
     * @return Integer
     */
    function countSquares($matrix) {

    }

}

```

Dart:

```
class Solution {  
  int countSquares(List<List<int>> matrix) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def countSquares(matrix: Array[Array[Int]]): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec count_squares(matrix :: [[integer]]) :: integer  
  def count_squares(matrix) do  
  
  end  
end
```

Erlang:

```
-spec count_squares(Matrix :: [[integer()]]) -> integer().  
count_squares(Matrix) ->  
.
```

Racket:

```
(define/contract (count-squares matrix)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 countSquares(vector<vector<int>>& matrix) {

    }
};

```

Java Solution:

```

/**
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 countSquares(int[][] matrix) {

    }
}

```

Python3 Solution:

```

"""
Problem: Count Square Submatrices with All Ones
Difficulty: Medium
Tags: array, dp

```

```

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 countSquares(self, matrix: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def countSquares(self, matrix):
"""
:type matrix: List[List[int]]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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[][]} matrix
 * @return {number}
 */
var countSquares = function(matrix) {

};

```

TypeScript Solution:

```

/**
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 countSquares(matrix: number[][]): number {

};

```

C# Solution:

```

/*
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 CountSquares(int[][] matrix) {

    }
}

```

C Solution:

```

/*
 * Problem: Count Square Submatrices with All Ones
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 countSquares(int** matrix, int matrixSize, int* matrixColSize) {

}

```

Go Solution:

```

// Problem: Count Square Submatrices with All Ones
// Difficulty: Medium
// Tags: array, dp
//
// 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 countSquares(matrix [][]int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun countSquares(matrix: Array<IntArray>): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func countSquares(_ matrix: [[Int]]) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Count Square Submatrices with All Ones
// Difficulty: Medium
// Tags: array, dp

```

```
//
// 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 count_squares(matrix: Vec<Vec<i32>>) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer}
def count_squares(matrix)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $matrix
     * @return Integer
     */
    function countSquares($matrix) {

    }

}
```

Dart Solution:

```
class Solution {
    int countSquares(List<List<int>> matrix) {

    }
}
```

Scala Solution:

```
object Solution {  
  def countSquares(matrix: Array[Array[Int]]): Int = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec count_squares(matrix :: [[integer]]) :: integer  
  def count_squares(matrix) do  
  
  end  
end
```

Erlang Solution:

```
-spec count_squares(Matrix :: [[integer()]]) -> integer().  
count_squares(Matrix) ->  
.
```

Racket Solution:

```
(define/contract (count-squares matrix)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
)
```