Problem 1074: Number of Submatrices That Sum to Target

Problem Information

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

Problem Description

Given a

matrix

and a

target

, return the number of non-empty submatrices that sum to

target

.

A submatrix

x1, y1, x2, y2

is the set of all cells

matrix[x][y]

with

x1 <= x <= x2

and

y1 <= y <= y2

.

Two submatrices

(x1, y1, x2, y2)

and

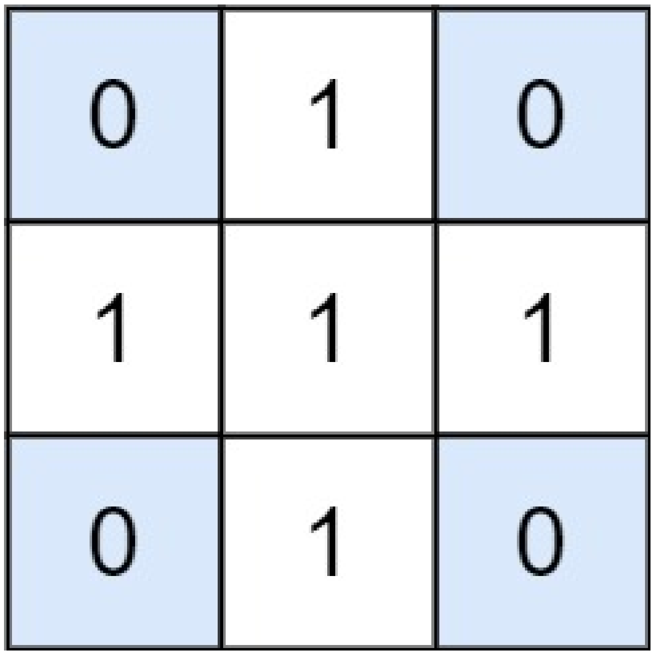
(x1', y1', x2', y2')

are different if they have some coordinate that is different: for example, if

x1 != x1'

.

Example 1:



0	1	0
1	1	1
0	1	0

Input:

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

Output:

4

Explanation:

The four 1x1 submatrices that only contain 0.

Example 2:

Input:

matrix = [[1,-1],[-1,1]], target = 0

Output:

5

Explanation:

The two 1x2 submatrices, plus the two 2x1 submatrices, plus the 2x2 submatrix.

Example 3:

Input:

matrix = [[904]], target = 0

Output:

0

Constraints:

1 <= matrix.length <= 100

1 <= matrix[0].length <= 100

-1000 <= matrix[i][j] <= 1000

-10^8 <= target <= 10^8

Code Snippets

C++:

```
class Solution {
public:
int numSubmatrixSumTarget(vector<vector<int>>& matrix, int target) {

}
};
```

Java:

```
class Solution {
public int numSubmatrixSumTarget(int[][] matrix, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numSubmatrixSumTarget(matrix: number[][], target: number): number {

};
```

C#:

```
public class Solution {
public int NumSubmatrixSumTarget(int[][] matrix, int target) {

}
}
```

C:

```
int numSubmatrixSumTarget(int** matrix, int matrixSize, int* matrixColSize,
int target) {

}
```

Go:

```
func numSubmatrixSumTarget(matrix [][]int, target int) int {

}
```

Kotlin:

```
class Solution {
fun numSubmatrixSumTarget(matrix: Array<IntArray>, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func numSubmatrixSumTarget(_ matrix: [[Int]], _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_submatrix_sum_target(matrix: Vec<Vec<i32>>, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Integer}
def num_submatrix_sum_target(matrix, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Integer
*/
function numSubmatrixSumTarget($matrix, $target) {

}
}
```

Dart:

```
class Solution {
int numSubmatrixSumTarget(List<List<int>> matrix, int target) {



}
}
```

Scala:

```
object Solution {
def numSubmatrixSumTarget(matrix: Array[Array[Int]], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_submatrix_sum_target(matrix :: [[integer]], target :: integer) ::
integer
def num_submatrix_sum_target(matrix, target) do

end
end
```

Erlang:

```
-spec num_submatrix_sum_target(Matrix :: [[integer()]], Target :: integer())
-> integer().
num_submatrix_sum_target(Matrix, Target) ->
.
```

Racket:

```
(define/contract (num-submatrix-sum-target matrix target)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int numSubmatrixSumTarget(vector<vector<int>>& matrix, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int numSubmatrixSumTarget(int[][] matrix, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Submatrices That Sum to Target
Difficulty: Hard
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def numSubmatrixSumTarget(self, matrix: List[List[int]], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} matrix
* @param {number} target
* @return {number}
*/
var numSubmatrixSumTarget = function(matrix, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function numSubmatrixSumTarget(matrix: number[][], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int NumSubmatrixSumTarget(int[][] matrix, int target) {

}
}
```

C Solution:

```
/*
* Problem: Number of Submatrices That Sum to Target
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int numSubmatrixSumTarget(int** matrix, int matrixSize, int* matrixColSize,
int target) {

}
```

Go Solution:

```
// Problem: Number of Submatrices That Sum to Target
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func numSubmatrixSumTarget(matrix [][]int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSubmatrixSumTarget(matrix: Array<IntArray>, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSubmatrixSumTarget(_ matrix: [[Int]], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Submatrices That Sum to Target
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn num_submatrix_sum_target(matrix: Vec<Vec<i32>>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Integer}
def num_submatrix_sum_target(matrix, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Integer
*/
function numSubmatrixSumTarget($matrix, $target) {

}
}
```

Dart Solution:

```
class Solution {
int numSubmatrixSumTarget(List<List<int>> matrix, int target) {
}
}
```

Scala Solution:

```
object Solution {
def numSubmatrixSumTarget(matrix: Array[Array[Int]], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_submatrix_sum_target(matrix :: [[integer]], target :: integer) ::
integer
def num_submatrix_sum_target(matrix, target) do

end
end
```

Erlang Solution:

```
-spec num_submatrix_sum_target(Matrix :: [[integer()]], Target :: integer())
-> integer().
num_submatrix_sum_target(Matrix, Target) ->
.
```

Racket Solution:

```
(define/contract (num-submatrix-sum-target matrix target)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```