Problem 2536: Increment Submatrices by One

Problem Information

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

Problem Description

You are given a positive integer

n

, indicating that we initially have an

n x n

0-indexed

integer matrix

mat

filled with zeroes.

You are also given a 2D integer array

query

. For each

query[i] = [row1

i

, col1

i

, row2

i

, col2

i

]

, you should do the following operation:

Add

1

to

every element

in the submatrix with the

top left

corner

(row1

i

, col1

i

)

and the

bottom right

corner

(row2

i

, col2

i

)

. That is, add

1

to

mat[x][y]

for all

row1

i

<= x <= row2

i

and

col1

i

<= y <= col2

i

.

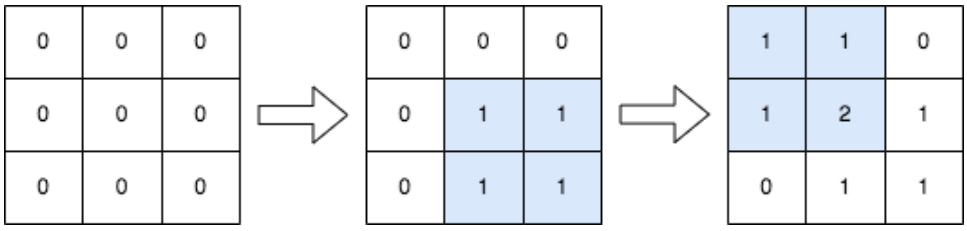
Return

the matrix

mat

after performing every query.

Example 1:



Input:

n = 3, queries = [[1,1,2,2],[0,0,1,1]]

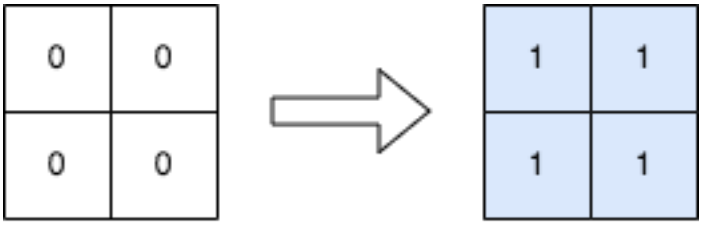
Output:

[[1,1,0],[1,2,1],[0,1,1]]

Explanation:

The diagram above shows the initial matrix, the matrix after the first query, and the matrix after the second query. - In the first query, we add 1 to every element in the submatrix with the top left corner (1, 1) and bottom right corner (2, 2). - In the second query, we add 1 to every element in the submatrix with the top left corner (0, 0) and bottom right corner (1, 1).

Example 2:



Input:

n = 2, queries = [[0,0,1,1]]

Output:

[[1,1],[1,1]]

Explanation:

The diagram above shows the initial matrix and the matrix after the first query. - In the first query we add 1 to every element in the matrix.

Constraints:

1 <= n <= 500

1 <= queries.length <= 10

4

0 <= row1

i

<= row2

i

< n

0 <= col1

i

<= col2

i

< n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> rangeAddQueries(int n, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[][] rangeAddQueries(int n, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def rangeAddQueries(self, n: int, queries: List[List[int]]) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def rangeAddQueries(self, n, queries):
"""
:type n: int
:type queries: List[List[int]]
:rtype: List[List[int]]
```

```
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} queries
* @return {number[][]}
*/
var rangeAddQueries = function(n, queries) {

};
```

TypeScript:

```
function rangeAddQueries(n: number, queries: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] RangeAddQueries(int n, int[][] queries) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** rangeAddQueries(int n, int** queries, int queriesSize, int*
queriesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func rangeAddQueries(n int, queries [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun rangeAddQueries(n: Int, queries: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func rangeAddQueries(_ n: Int, _ queries: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn range_add_queries(n: i32, queries: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[][]}
def range_add_queries(n, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
```

```
* @param Integer[][] $queries
* @return Integer[][]
*/
function rangeAddQueries($n, $queries) {

}
}
```

Dart:

```
class Solution {
List<List<int>> rangeAddQueries(int n, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def rangeAddQueries(n: Int, queries: Array[Array[Int]]): Array[Array[Int]] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec range_add_queries(n :: integer, queries :: [[integer]]) :: [[integer]]
def range_add_queries(n, queries) do

end
end
```

Erlang:

```
-spec range_add_queries(N :: integer(), Queries :: [[integer()]]) ->
[[integer()]].
range_add_queries(N, Queries) ->
.
```

Racket:

```
(define/contract (range-add-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Increment Submatrices by One
* 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:
vector<vector<int>> rangeAddQueries(int n, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Increment Submatrices by One
* 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[][] rangeAddQueries(int n, int[][] queries) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Increment Submatrices by One
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 rangeAddQueries(self, n: int, queries: List[List[int]]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rangeAddQueries(self, n, queries):
"""
:type n: int
:type queries: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Increment Submatrices by One
* 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} n
* @param {number[][]} queries
* @return {number[][]}
*/
var rangeAddQueries = function(n, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Increment Submatrices by One
* 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 rangeAddQueries(n: number, queries: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Increment Submatrices by One
* 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[][] RangeAddQueries(int n, int[][] queries) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Increment Submatrices by One
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** rangeAddQueries(int n, int** queries, int queriesSize, int*
queriesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Increment Submatrices by One
// 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 rangeAddQueries(n int, queries [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun rangeAddQueries(n: Int, queries: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func rangeAddQueries(_ n: Int, _ queries: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Increment Submatrices by One
// 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 range_add_queries(n: i32, queries: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[][]}
def range_add_queries(n, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $queries
* @return Integer[][]
*/
function rangeAddQueries($n, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> rangeAddQueries(int n, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def rangeAddQueries(n: Int, queries: Array[Array[Int]]): Array[Array[Int]] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec range_add_queries(n :: integer, queries :: [[integer]]) :: [[integer]]
def range_add_queries(n, queries) do

end
end
```

Erlang Solution:

```
-spec range_add_queries(N :: integer(), Queries :: [[integer()]]) ->
[[integer()]].
```

```
range_add_queries(N, Queries) ->
.
```

Racket Solution:

```
(define/contract (range-add-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```