# Problem 2718: Sum of Matrix After Queries

## Problem Information

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

## Problem Description

You are given an integer

n

and a

0-indexed

2D array

queries

where

queries[i] = [type

i

, index

i

, val

i

]

.

Initially, there is a

0-indexed

n x n

matrix filled with

0

's. For each query, you must apply one of the following changes:

if

type

i

== 0

, set the values in the row with

index

i

to

val

i

, overwriting any previous values.

if

type

i

== 1

, set the values in the column with

index

i

to

val

i

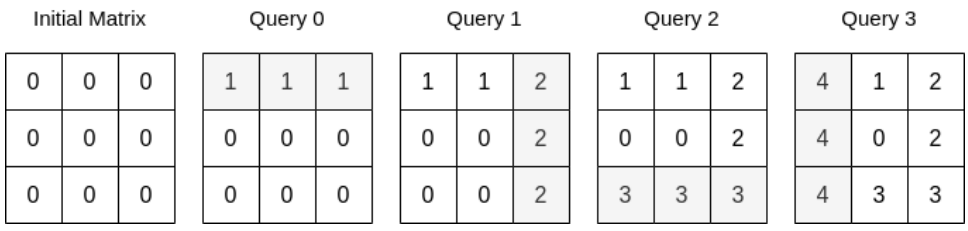
, overwriting any previous values.

Return

the sum of integers in the matrix after all queries are applied

.

Example 1:




Input:

n = 3, queries = [[0,0,1],[1,2,2],[0,2,3],[1,0,4]]

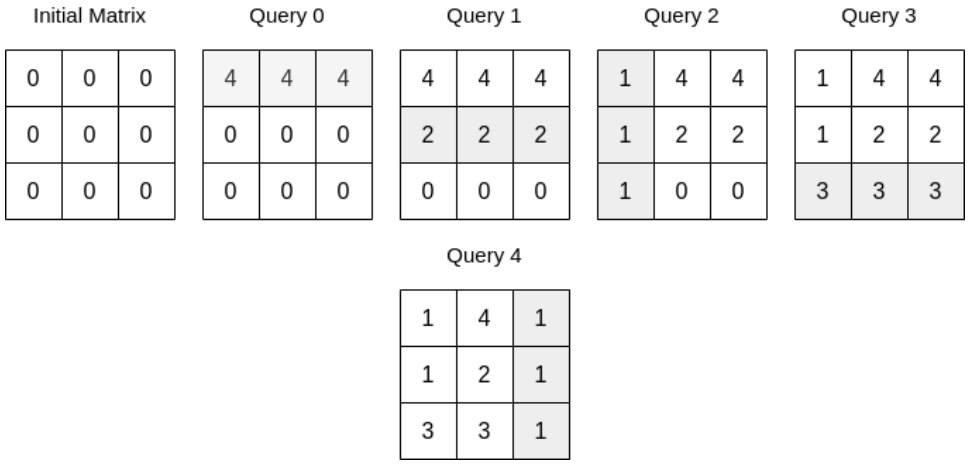
Output:

23

Explanation:

The image above describes the matrix after each query. The sum of the matrix after all queries are applied is 23.

Example 2:




Input:

n = 3, queries = [[0,0,4],[0,1,2],[1,0,1],[0,2,3],[1,2,1]]

Output:

17

Explanation:

The image above describes the matrix after each query. The sum of the matrix after all queries are applied is 17.

Constraints:

1 <= n <= 10

4

1 <= queries.length <= 5 * 10

4

queries[i].length == 3

0 <= type

i

<= 1

0 <= index

i

< n

0 <= val

i

<= 10

5

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

```
}
}
```

**C:**

```
long long matrixSumQueries(int n, int** queries, int queriesSize, int*
queriesColSize) {

}
```

**Go:**

```
func matrixSumQueries(n int, queries [][]int) int64 {

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

```
impl Solution {
pub fn matrix_sum_queries(n: i32, queries: Vec<Vec<i32>>) -> i64 {

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

```
end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (matrix-sum-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
* 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:
long long matrixSumQueries(int n, vector<vector<int>>& queries) {

}
};
```

### Java Solution:

```
/**
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
* 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 long matrixSumQueries(int n, int[][] queries) {

}
}
```

### Python3 Solution:

```
"""
Problem: Sum of Matrix After Queries
Difficulty: Medium
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 matrixSumQueries(self, n: int, queries: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
* 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} n
* @param {number[][]} queries
* @return {number}
*/
var matrixSumQueries = function(n, queries) {

};
```

### TypeScript Solution:

```
/**
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
* 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 matrixSumQueries(n: number, queries: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
```

```
* 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 long MatrixSumQueries(int n, int[][] queries) {

}
}
```

### C Solution:

```
/*
* Problem: Sum of Matrix After Queries
* Difficulty: Medium
* 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
*/

long long matrixSumQueries(int n, int** queries, int queriesSize, int*
queriesColSize) {

}
```

### Go Solution:

```
// Problem: Sum of Matrix After Queries
// Difficulty: Medium
// 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 matrixSumQueries(n int, queries [][]int) int64 {
```

```
}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Sum of Matrix After Queries
// Difficulty: Medium
// 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 matrix_sum_queries(n: i32, queries: Vec<Vec<i32>>) -> i64 {

}
}
```

### Ruby Solution:

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

```
end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (matrix-sum-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```