



Problem Description

Math Simulation

This LeetCode problem presents an m x n matrix initially filled with all zeroes. We're given a list of indices where each element is a pair $[r_i, c_i]$, indicating a specific row r_i and column c_i in the matrix. For each pair of indices, we need to perform two operations:

1. Increment all the cells in the specified row r_i.

the problem to a matter of counting increments per row and column.

2. Increment all the cells in the specified column c_i.

Our goal is to figure out how many cells in the matrix will have odd values after performing all the specified row and column increments.

Intuition

find out how many cells will be odd. Instead, we can track the increment counts separately for rows and columns. We create two lists, row of size m and col of size n, to count how many times each row and each column is incremented. When we

The intuition behind the solution lies in recognizing that we don't actually need to perform all the increments on the entire matrix to

process the indices, we increment the corresponding count in row and col arrays for each given pair [r_i, c_i]. After all increments have been tallied, the value of a cell (i, j) in the matrix will be the sum of increments for its row (i.e., row[i]) and

its column (i.e., col[j]). If the sum of increments for a cell is odd, then the cell value will be odd. Therefore, the cell will contribute to our final count of odd-valued cells in the matrix.

combination of row[i] and col[j] increments and checking if the result is odd ((i + j) % 2) being 1). This approach avoids the need for a potentially expensive operation of incrementing every cell in the matrix and instead simplifies

times each row and column is incremented rather than updating the whole matrix directly. Here's a walkthrough of the

The code then calculates the total number of odd-valued cells using a simple list comprehension, adding up each possible

Solution Approach

The solution uses a straightforward and efficient approach by utilizing additional data structures to keep track of the number of

implementation steps: 1. Initialize two arrays row and col with sizes corresponding to the number of rows m and the number of columns n in the matrix, respectively. These will hold the number of times each row and column needs to be incremented.

1 row = [0] * m2 col = [0] * n

2. Iterate through each given index in the indices array. For each pair [r_i, c_i], increment the value at position r_i in the row array

and the value at position c_i in the col array by 1. This step effectively counts the number of times each row and column should

```
be incremented without actually altering the matrix.
```

1 for r, c in indices:

col[c] += 13. Calculate the number of odd-valued cells. A cell at position (i, j) will have an odd value if the sum of increments from its row and column is odd. To find this, we use a list comprehension that iterates over every possible combination of row and column

increments (row[i] + col[j]), and if the sum is odd ((i + j) % 2 == 1), it contributes to the count.

```
By running through all combinations of row and column increments, the algorithm efficiently calculates the final count of odd-valued
```

1 return sum((i + j) % 2 for i in row for j in col)

updating the entire matrix after each increment, thereby offering an optimized way to solve the problem. Example Walkthrough

cells after performing all specified operations. This solution avoids the time and space complexity issues that would arise from

approach would work with these inputs:

1. Initialize two arrays row and col to keep track of the increments: 1 row = [0, 0, 0] # For 3 rows

Let's consider a small example with a 3x4 matrix (m=3, n=4) and given indices [[0, 1], [1, 1], [2, 2]]. Here's how the solution

2. Process the given list of indices:

```
1 indices = [[0, 1], [1, 1], [2, 2]]
2 for r, c in indices:
  row[r] += 1
```

2 col = [0, 0, 0, 0] # For 4 columns

6 # After processing the indices, we have:

4 # Calculate the count:

8 # Total odd value count is 4.

col[c] += 1

```
8 col = [0, 2, 1, 0] # Column 1 is incremented twice and Column 2 once
  Our row and col arrays now represent the total number of times each row and column has been incremented.
3. Determine the odd-valued cells in the matrix:
   1 # A cell (i, j) will be odd if (row[i] + col[j]) is odd.
   2 odd_count = sum((row[i] + col[j]) % 2 == 1 for i in range(3) for j in range(4))
```

 $7 \text{ row} = [1, 1, 1] \# Row \emptyset, Row 1, and Row 2 are each incremented once$

10 return odd_count

7 # For i=2 (Row 2): (1+0) % 2, (1+2) % 2, (1+1) % 2, (1+0) % 2 => 2 odd values (at columns 1 and 2)

5 # For i=0 (Row 0): (1+0) % 2, (1+2) % 2, (1+1) % 2, (1+0) % 2 => 1 odd value (at column 1)

6 # For i=1 (Row 1): (1+0) % 2, (1+2) % 2, (1+1) % 2, (1+0) % 2 => 1 odd value (at column 1)

So, the final count of cells with odd values is 4 for the given example. This method efficiently calculates the odd cell count without

 $col_counts = [0] * cols$

odd_values_count = sum(

```
modifying the entire matrix, but simply by tracking the number of increments in each row and column.
Python Solution
   class Solution:
       def oddCells(self, rows: int, cols: int, indices: List[List[int]]) -> int:
           # Initialize two arrays to track the counts of increments in each row and column,
           # respectively. Initially, all cells have even counts (0).
           row_counts = [0] * rows
```

for r, c in indices: 10 row_counts[r] += 1 11 12 col_counts[c] += 1 13

For each cell, the value is the sum of the row count and column count.

```
20
               for col_count in col_counts
21
22
23
           # Return the total count of cells with odd values.
24
           return odd_values_count
25
Java Solution
```

// Create arrays to keep track of the increments for rows and columns

// Increment the corresponding row and column counters for each index pair

Loop through each pair of indices representing [row, column].

Increment the corresponding row and column count.

A cell has an odd value if the sum is an odd number.

Compute the number of cells with odd values.

public int oddCells(int m, int n, int[][] indices) {

// for the corresponding row and column is odd

int[] rowCount = new int[m];

int[] colCount = new int[n];

rowCount[rowIndex]++;

colCount[colIndex]++;

for (int i = 0; i < m; i++) {

for (int[] indexPair : indices) {

int rowIndex = indexPair[0];

int colIndex = indexPair[1];

(row_count + col_count) % 2

for row_count in row_counts

// Initialize a counter for the number of cells with odd values 15 int oddValueCount = 0; 16 18 // Iterate over each cell to determine if the sum of increments

class Solution {

14

15

16

17

18

19

12

13

14

19

20

21

```
for (int j = 0; j < n; j++) {
                    if ((rowCount[i] + colCount[j]) % 2 != 0) {
22
                       // Increment the count if the cell value is odd
                        oddValueCount++;
24
25
26
27
28
29
           // Return the total count of cells with odd values
           return oddValueCount;
30
31
32 }
33
C++ Solution
 1 #include <vector>
 2 using namespace std;
   class Solution {
 5 public:
       // Function to count how many cells will have odd values after performing the operations indicated in 'indices'
       int oddCells(int numberOfRows, int numberOfColumns, vector<vector<int>>& indices) {
           vector<int> rows(numberOfRows, 0); // Create a vector to keep track of increments in each row
           vector<int> cols(numberOfColumns, 0); // Create a vector to keep track of increments in each column
10
           // Iterate over each pair of indices and increment the relevant rows and columns
11
           for (const vector<int>& indexPair : indices) {
               int rowIndex = indexPair[0];
13
               int colIndex = indexPair[1];
14
15
               rows[rowIndex]++;
               cols[colIndex]++;
16
17
18
           int oddCount = 0; // Initialize a counter for cells with odd values
19
```

// Nested loop to count the cells that will have odd values

for (int j = 0; j < numberOfColumns; ++j) {</pre>

if ((rows[i] + cols[j]) % 2 != 0) {

proportional to the number of rows m and columns n, respectively.

for (int i = 0; i < numberOfRows; ++i) {</pre>

oddCount++;

30 31 // Return the total count of cells with odd values 32 return oddCount; 33

20

21

22

23

24

26

27

28

29

```
34 };
35
Typescript Solution
1 // Importing the 'Array' type from TypeScript for type definitions
   import { Array } from "typescript";
   // Function to count how many cells will have odd values after performing the operations indicated in 'indices'
   function oddCells(numberOfRows: number, numberOfColumns: number, indices: Array<Array<number>>): number {
       // Create an array to keep track of increments in each row
       let rows: Array<number> = new Array(numberOfRows).fill(0);
       // Create an array to keep track of increments in each column
       let cols: Array<number> = new Array(numberOfColumns).fill(0);
10
       // Iterate over each pair of indices and increment the respective rows and columns
       indices.forEach(indexPair => {
           let rowIndex: number = indexPair[0];
           let colIndex: number = indexPair[1];
14
           rows[rowIndex]++;
15
           cols[colIndex]++;
16
       });
17
       // Initialize a counter for cells with odd values
19
20
       let oddCount: number = 0;
21
22
       // Nested loop to count the cells that will have odd values
       for (let i = 0; i < numberOfRows; i++) {</pre>
23
           for (let j = 0; j < numberOfColumns; j++) {</pre>
               // If the sum of the increments for the current cell's row and column is odd, increment oddCount
               if ((rows[i] + cols[j]) % 2 !== 0) {
                   oddCount++;
```

// If the sum of the increments for the current cell's row and column is odd, increment oddCount

26 27 28 29 30 31 32 // Return the total count of cells with odd values return oddCount; 33 34 } 35

Time and Space Complexity

two gives the final time complexity. The space complexity of the code is 0(m + n), as it requires additional space to store the row and col arrays, whose sizes are

by a nested loop that iterates through all possible combinations of rows and columns, taking 0(m * n) time. The addition of these

The time complexity of the provided code is 0(m * n + k), where m is the number of rows, n is the number of columns, and k is the

length of the indices list. This is because the code consists of a single loop through the indices list, which takes O(k) time, followed