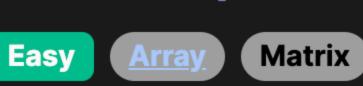
## 1582. Special Positions in a Binary Matrix



### **Problem Description**

In this LeetCode problem, we're given a binary matrix mat, which is made up of only 0s and 1s. Our task is to count how many 'special positions' are in the matrix. A special position is defined as one where the value at that position is 1 and all other values in the same row and column are 0. Here, the matrix is indexed starting at (0,0) for the top-left element.

Intuition

the corresponding counts for row i and column j are both exactly 1, then the position (i, j) is a special position. Here's the breakdown: 1. We initialize two lists, r and c, to keep track of the count of 1s in each row and each column, respectively.

The intuition behind the solution is to first count the number of 1s in each row and each column. If a position (i, j) has a 1, and

- 2. By double looping through the matrix, we update these counts. 3. After populating r and c, we go through the matrix again, checking if a 1 is in a position (i, j) such that r[i] and c[j] are both exactly 1.
- 4. If the condition from step 3 is met, we increment our ans variable, which holds the count of special positions.
- 5. We return the value of ans as the final result.
- Solution Approach

### includes the following steps:

Initialize Count Arrays: The code initializes two arrays r and c with lengths equal to the number of rows m and columns n of the input matrix, respectively. These arrays are used to keep track of the sum of 1s in each row and column, hence initialized to all 0s.

The solution approach follows a straightforward algorithmic pattern that is quite common in matrix-related problems, which

- **Populate Count Arrays:** The solution uses a nested loop where i iterates over the rows and j iterates over the columns. For each cell in the matrix, if the value mat[i][j] is 1, the sum in the corresponding count arrays r[i] and c[j] are incremented by 1. This allows us to accumulate the number of 1s for each row and each column in their respective counters.
- Identify Special Positions: With the populated count arrays, we loop through the matrix for the second time. During this iteration, we check if the value at mat[i][j] is 1 and if r[i] and c[j] are both equal to 1. This condition verifies that the current position (i, j) is special as it is the only 1 in its row and column.
- Count Special Positions: If the condition in the previous step is satisfied, we increment the variable ans which is used to count the number of special positions. Return the Result: Once the entire matrix has been scanned during the second iteration, the ans variable contains the total
- The data structures used are quite simple and effective; we are using two one-dimensional arrays (r for rows and c for columns) to keep the sums. The algorithmic pattern employed is also straightforward, involving iterations and condition checking. This approach is efficient since each element in the matrix is processed a constant number of times, resulting in a time complexity of

O(m\*n), where m and n are the number of rows and columns in the matrix, respectively. The space complexity is O(m+n), which is

required for the row and column sum arrays. **Example Walkthrough** Let's consider a 3×4 binary matrix mat for our example:

Following our algorithmic steps:

all 0s. So, r = [0, 0, 0] and c = [0, 0, 0, 0].

**Populate Count Arrays:** We iterate through the matrix mat:

[1, 0, 0, 0],

[0, 0, 1, 0],

 $\circ$  For i=0 (first row), we find mat [0] [0] is 1, so we increment r[0] and c[0] by 1. Now, r = [1, 0, 0] and c = [1, 0, 0, 0].  $\circ$  For i=1 (second row), we find mat[1][2] is 1, so we increment r[1] and c[2] by 1. Now, r = [1, 1, 0] and c = [1, 0, 1, 0].

 $\circ$  For i=2 (third row), we find mat [2] [1] is 1, so we increment r[2] and c[1] by 1. Now, r = [1, 1, 1] and c = [1, 1, 1, 0].

After populating the counts arrays, r and c now accurately reflect the number of 1s in each row and column.

Initialize Count Arrays: We initialize two arrays r with length 3 (number of rows) and c with length 4 (number of columns), to

 $\circ$  Check position (2,1). Since mat [2] [1] is 1 and both r[2] and c[1] are exactly 1, this is a special position as well. Every 1 we encountered is indeed in a special position.

count of special positions. This value is returned as the output.

ans would be 3. **Return the Result**: Our function would return ans, the total count of special positions, which in this case is 3.

**Identify Special Positions**: With the count arrays set up, we go through the matrix once more:

• Check position (0,0). Since mat [0] [0] is 1 and both r[0] and c[0] are exactly 1, this is a special position.

 $\circ$  Check position (1,2). Since mat [1] [2] is 1 and both r[1] and c[2] are exactly 1, this is also a special position.

special positions, as identified using the solution approach. The row and column counts help efficiently pinpoint the special positions in just two scans of the matrix. Solution Implementation

In this straightforward example, our methodical walk-through demonstrates that the provided binary matrix mat contains three

Count Special Positions: We increment our variable ans for each special position identified. As we found 3 special positions,

### # Initialize row\_sum and col\_sum to keep track of the sum of each row and column row\_sum = [0] \* num\_rows $col_sum = [0] * num_cols$

def numSpecial(self, mat: List[List[int]]) -> int:

num\_rows, num\_cols = len(mat), len(mat[0])

for j, value in enumerate(row):

specialCount++;

int numSpecial(vector<vector<int>>& mat) {

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

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

rowCount[i] += mat[i][j];

colCount[j] += mat[i][j];

return specialCount;

row\_sum[i] += value

col\_sum[j] += value

for i, row in enumerate(mat):

# Get the number of rows 'm' and columns 'n' of the matrix

# Calculate the sum of elements for each row and column

# Check for special positions where the value is 1

# and its row and column sums are both exactly 1

```
# Initialize variable 'special_count' to count special positions
special_count = 0
```

**Python** 

class Solution:

```
for i in range(num_rows):
            for j in range(num_cols):
                if mat[i][j] == 1 and row_sum[i] == 1 and col_sum[j] == 1:
                    # Increment the count of special positions
                    special_count += 1
        # Return the final count of special positions
        return special_count
Java
class Solution {
    public int numSpecial(int[][] mat) {
        int numRows = mat.length, numCols = mat[0].length;
        int[] rowCount = new int[numRows];
        int[] colCount = new int[numCols];
       // Calculate the sum of each row and each column
        for (int i = 0; i < numRows; ++i) {</pre>
            for (int j = 0; j < numCols; ++j) {</pre>
                rowCount[i] += mat[i][j];
                colCount[j] += mat[i][j];
        int specialCount = 0;
       // Iterate through the matrix to find special elements
        // A special element is defined as the element that is the only '1' in its row and column
        for (int i = 0; i < numRows; ++i) {
            for (int j = 0; j < numCols; ++j) {</pre>
                // Check if the current element is '1' and its corresponding
```

// row and column sums are '1' which would mean it's a special element

// A position (i, j) is called special if mat[i][j] is 1 and all other elements in row i and column j are 0.

if (mat[i][j] == 1 && rowCount[i] == 1 && colCount[j] == 1) {

// Return the total count of special elements found in the matrix

// Function to count the number of special positions in a binary matrix.

int numRows = mat.size(); // Number of rows in the matrix

int numCols = mat[0].size(); // Number of columns in the matrix

vector<int> rowCount(numRows, 0); // Row count to store the sum of each row

// Fill rowCount and colCount by summing the values in each row and column

vector<int> colCount(numCols, 0); // Column count to store the sum of each column

# C++

public:

class Solution {

```
int specialCount = 0; // Variable to store the number of special positions found
       // Search for special positions. A position (i, j) is special if
       // mat[i][j] is 1 and the sum of both row i and column j is 1.
        for (int i = 0; i < numRows; ++i) {
            for (int j = 0; j < numCols; ++j) {</pre>
                if (mat[i][j] == 1 && rowCount[i] == 1 && colCount[j] == 1) {
                    specialCount++; // Increment count if a special position is found
        return specialCount; // Return the total count of special positions
};
TypeScript
function countSpecialElements(matrix: number[][]): number {
    // Get the number of rows and columns from the matrix
    const rowCount = matrix.length;
    const colCount = matrix[0].length;
    // Create arrays to store the sum of elements in each row and column,
    // and initialize each element of the arrays to 0
    const rowSums = new Array(rowCount).fill(0);
    const colSums = new Array(colCount).fill(0);
    // First pass: Calculate the number of 1's in each row and column
    for (let rowIndex = 0; rowIndex < rowCount; rowIndex++) {</pre>
        for (let colIndex = 0; colIndex < colCount; colIndex++) {</pre>
            // If the element at the current position is 1, increment
            // the corresponding row and column sums
            if (matrix[rowIndex][colIndex] === 1) {
                rowSums[rowIndex]++;
                colSums[colIndex]++;
    // Initialize the result variable which will hold the count of special elements
    let specialCount = 0;
```

if (matrix[rowIndex][colIndex] === 1 && rowSums[rowIndex] === 1 && colSums[colIndex] === 1) {

```
def numSpecial(self, mat: List[List[int]]) -> int:
    # Get the number of rows 'm' and columns 'n' of the matrix
    num_rows, num_cols = len(mat), len(mat[0])
    # Initialize row_sum and col_sum to keep track of the sum of each row and column
    row sum = [0] * num rows
    col_sum = [0] * num_cols
    # Calculate the sum of elements for each row and column
    for i, row in enumerate(mat):
        for j, value in enumerate(row):
            row_sum[i] += value
            col_sum[j] += value
    # Initialize variable 'special_count' to count special positions
    special_count = 0
    # Check for special positions where the value is 1
    # and its row and column sums are both exactly 1
    for i in range(num_rows):
        for j in range(num_cols):
```

if mat[i][j] == 1 and row\_sum[i] == 1 and col\_sum[j] == 1:

# Increment the count of special positions

special\_count += 1

# Return the final count of special positions

// Second pass: Check for special elements, which are the elements

for (let colIndex = 0; colIndex < colCount; colIndex++) {</pre>

// Check if the current element is 1 and if it's the only one

// in its row and column, if so increment the specialCount

for (let rowIndex = 0; rowIndex < rowCount; rowIndex++) {</pre>

// that are the only 1 in their row and column

specialCount++;

// Return the count of special elements

return specialCount;

class Solution:

## The time complexity of the code can be analyzed by looking at the number of nested loops and the operations within them.

**Time Complexity** 

once.

return special\_count

Time and Space Complexity

• The code first initializes the row and column count arrays r and c, which is O(m) and O(n) respectively, where m is the number of rows and n is the number of columns in the input mat. • The first nested for loop iterates through all elements of the matrix to populate r and c, which will be 0(m \* n) since every element is visited

- The second nested for loop also iterates through the entire matrix to count the number of special elements based on the conditions that rely on the previous computations stored in r and c. This is also 0(m \* n).
- Hence, the overall time complexity is 0(m \* n) because this dominates the overall performance of the code.

**Space Complexity** The space complexity of the code includes the space used for the input and any auxiliary space used:

• The input matrix itself does not count towards auxiliary space complexity as it is given.

• Two arrays r and c of length m and n are created to keep track of the sum of each row and column, which gives us 0(m + n). Therefore, the auxiliary space complexity is 0(m + n).