



Problem Description

Given a matrix where each cell is either a 0 or a 1, we need to identify which row contains the most 1s. The matrix is formatted as a

list of lists in  $m \times n$  dimensions, where m represents the number of rows and n the number of columns.

Our goal is to find the row index (0-indexed) with the highest number of 1s. If there are multiple rows with the same maximum count of 1s, we prioritize the row with the smallest index.

For the output, we must return a list with two elements: the index of the row containing the maximum count of ones and the count of ones in that respective row.

## Intuition

To arrive at the solution, we can iterate through each row of the matrix and count the number of 1s in that row. We do this by using Python's built-in count method on lists, which efficiently returns the number of occurrences of a particular element—in this case, the numeral 1.

represents the count of 1s.

We initialize a list ans with two elements [0, 0]. The first element represents the index of the row and the second element

As we iterate over each row:

- We count the 1s,
- 2. We compare this count with the current maximum (stored in ans [1]), 3. If the count is higher, we update ans with the current row index and the new maximum count.
- By scanning through the matrix once, we find the required row index and the number of 1s in that row. At the end of iteration, ans will

contain the index of the desired row and the highest count of ones found.

Solution Approach

The implementation of the solution uses a straightforward algorithm that takes advantage of simple iteration and Python's built-in list operations to solve the problem efficiently.

answer list with the current row index and the count:

Here is a step-by-step guide to the algorithm:

- 1. Initialize an answer list ans = [0, 0]. The first element is the row index (initialized to 0) and the second element is the count of 1s (also initialized to 0).
- function. 1 for i, row in enumerate(mat):

2. Iterate over each row in the matrix with a loop structure, keeping track of the current row index using Python's enumerate

```
3. Within the loop, count the number of 1s in the current row. This is done by using the count method on the row list:
```

1 cnt = row.count(1)

```
4. Check if the current row's 1s count (cnt) is greater than the maximum count of 1s found so far (ans [1]). If it is, then update the
```

1 if ans[1] < cnt: ans = [i, cnt]

final output. 1 return ans

5. Once all rows have been checked, the list ans contains the required index and the count of 1s. This list is then returned as the

since no extra space is used proportionally to the size of the input, aside from the space needed to store the answer ans. Example Walkthrough

By using this approach, the solution minimizes complexity by only passing through the matrix a single time, which gives it a time

complexity of O(m\*n), where m is the number of rows and n is the number of columns in the matrix. The space complexity is O(1)

# Let's walk through an example to illustrate the solution approach. Imagine we have the following matrix:

1 matrix = [

1. Initialize an answer list ans = [0, 0]. This list will hold the index of the row with the maximum 1s and the count of 1s in that respective row.

We need to identify which row has the most 1s.

- 2. Start iterating over each row in the matrix using enumerate:
- ans remains [0, 0] because 2 (current row's 1s count) is not greater than 0 (maximum 1s count so far). For the second row (i=1), the count of 1s is 3.
  - For the third row (i=2), the count of 1s is 2.

For the first row (i=0), the count of 1s is 2.

- Update ans to [1, 3] because 3 is greater than the maximum 1s count so far, which was 0.
- ans remains [1, 3] because 2 is not greater than 3 (current maximum 1s count). The fourth row (i=3) has a count of 1s as 0.

# Enumerate over the rows of the matrix.

public int[] rowAndMaximumOnes(int[][] mat) {

// Count the number of ones in the current row

result[0] = rowIndex; // Update row index

result[1] = onesCount; // Update max count of ones

for (int element : matrix[rowIndex]) {

onesCount += element == 1;

if (onesCount > maxOnes) {

- ans remains [1, 3] since 0 is not greater than 3. 3. Now that all rows have been checked, ans contains the index of the row with the maximum number of 1s and the count itself,
- which is [1, 3]. Thus, the output is [1, 3], meaning the second row (0-indexed) has the most number of 1s, which are 3 in total.
- **Python Solution**

def row\_and\_maximum\_ones(self, matrix: List[List[int]]) -> List[int]: # Initialize a variable to keep track of the row index # with the maximum number of ones, and the number of ones in that row. result = [0, 0]

int[] result = new int[2]; // Holds the index of the row and maximum number of ones.

for (int rowIndex = 0; rowIndex < mat.length; rowIndex++) { // Iterate over each row in the matrix.

### for i, row in enumerate(matrix): # Count the number of ones in the current row. 9 ones\_count = row.count(1) 10

class Solution:

```
11
12
               # If the count of ones in the current row is greater than
13
               # the maximum found so far, update the result.
               if result[1] < ones_count:</pre>
14
                   result = [i, ones_count] # Store the current row index and the count of ones.
15
16
17
           return result # Return the result list containing the row index and the maximum count of ones.
18
Java Solution
   class Solution {
       /**
        * This method finds the row with the maximum number of ones in a binary matrix.
        * @param mat A binary matrix of integers where each integer is either 0 or 1.
        * @return An array containing the index of the row with the maximum number of ones
                  and the count of ones in that row.
```

## int onesCount = 0; // Counter for number of ones in the current row. 14

8

10

11

12

```
for (int cellValue : mat[rowIndex]) { // Iterate over each element in the current row.
15
                    if (cellValue == 1) { // Check if the current element is a one.
16
17
                        onesCount++; // Increment the ones counter.
18
19
20
21
               // Update result if the current row has more ones than previously found.
22
               if (result[1] < onesCount) {</pre>
23
                    result[0] = rowIndex; // Set the index of the row with the most ones.
24
                    result[1] = onesCount; // Set the new maximum number of ones.
25
26
27
28
            return result; // Return the array containing the row index and maximum number of ones.
29
30 }
31
C++ Solution
 1 class Solution {
2 public:
       // Function to find the row with the maximum number of ones and return that row index and count of ones in it
       vector<int> rowAndMaximumOnes(vector<vector<int>>& matrix) {
            vector<int> result(2); // Initialize result vector to store row index and max count of ones
            int maxOnes = 0; // Variable to store the current maximum number of ones
           // Iterate through each row in the matrix
           for (int rowIndex = 0; rowIndex < matrix.size(); ++rowIndex) {</pre>
                int onesCount = 0; // Variable to count the ones in the current row
11
```

### 23 24 25 // Return the result vector containing the index of the row and the maximum number of ones 26

12

13

14

15

16 17

18

19

20

21

22

22

24

23 }

```
return result;
27
28 };
29
Typescript Solution
1 // Function to find the row with the maximum number of ones in a binary matrix
2 // and return an array containing the row index and the count of ones.
   function rowAndMaximumOnes(mat: number[][]): number[] {
       // Initialize the answer array with the first element as the row index
       // and the second element as the count of ones.
       const answer: number[] = [0, 0];
       // Iterate through each row of the matrix
       for (let rowIndex = 0; rowIndex < mat.length; rowIndex++) {</pre>
           // Count the number of ones in the current row by summing up the values
10
           const onesCount = mat[rowIndex].reduce((total, value) => total + value, 0);
11
12
13
           // If the count of ones in the current row is greater than the current maximum,
           // update the answer array with the new row index and ones count
14
           if (answer[1] < onesCount) {</pre>
               answer[0] = rowIndex;
16
               answer[1] = onesCount;
17
18
19
20
       // Return the answer array with the row index and count of ones for the row with the maximum number of ones
21
```

// If the current row has more ones than the previous maximum, update the result

maxOnes = onesCount; // Update max ones for comparison in subsequent iterations

return answer:

Time and Space Complexity The time complexity of the given code is 0(m\*n), where m is the number of rows and n is the number of columns in the matrix mat. This is because the code iterates through each row with enumerate(mat) and counts the number of 1's in that row with row.count(1),

which requires traversing the entire row. The space complexity of the given code is 0(1). The additional space used by the algorithm is constant and does not scale with the

input size since only a fixed-size list ans of length 2 is used to store the result and no other additional data structures are allocated.