Sum of elements in a matrix

School Accuracy: 74.5% Submissions: 26K+ Points: 0

Given a non null integer matrix Grid of dimensions NxM.Calculate the sum of its elements.

Example 1:

Input:
N=2,M=3
Grid=
[[1,0,1],
[-8,9,-2]]
Output:
1
Explanation:
The sum of all elements of the matrix is (1+0+1-8+9-2)=1.

Example 2:

Input:
N=3,M=5
Grid=
[[1,0,1,0,1],
[0,1,0,1,0],
[-1,-1,-1,-1,-1]]
Output:
0
Explanation:
The sum of all elements of the matrix are (1+0+1+0+1+0+1+0+1+0-1-1-1-1-1)=0.

Your Task:

You don't need to read input or print anything.Your task is to complete the function **sumOfMatrix()** which takes two integers N ,M and a 2D array Grid as input parameters and returns the sum of all the elements of the Grid.

Expected Time Complexity:O(N*M)

Expected Auxillary Space:O(1)

Constraints:

1<=N,M<=1000

-1000<=Grid[i][j]<=1000

```cpp
// https://www.geeksforgeeks.org/problems/sum-of-elements-in-a-matrix2000/1
// GFG Sum of elements in a matrix

/*
    initialize sum =0
    iterate through each element and accumulate the sum

*/

 class Solution {
   public:
     int sumOfMatrix(int N, int M, vector<vector<int>> Grid) {
         int sum=0;
         for(int i=0;i<N;i++){
             for(int j=0;j<M;j++){
                 sum+=Grid[i][j];
             }
         }
         return sum;
     }
 };
```

1672. Richest Customer Wealth

Solved

Easy | Topics | Companies | Hint

You are given an `m x n` integer grid `accounts` where `accounts[i][j]` is the amount of money the i^{th} customer has in the j^{th} bank. Return *the **wealth** that the richest customer has.*

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

Example 1:

```
Input: accounts = [[1,2,3],[3,2,1]]
Output: 6
Explanation:
1st customer has wealth = 1 + 2 + 3 = 6
2nd customer has wealth = 3 + 2 + 1 = 6
Both customers are considered the richest with a wealth of 6 each, so return 6.
```

Example 2:

```
Input: accounts = [[1,5],[7,3],[3,5]]
Output: 10
Explanation:
1st customer has wealth = 6
2nd customer has wealth = 10
3rd customer has wealth = 8
The 2nd customer is the richest with a wealth of 10.
```

Example 3:

```
Input: accounts = [[2,8,7],[7,1,3],[1,9,5]]
Output: 17
```

Constraints:

- `m == accounts.length`
- `n == accounts[i].length`
- `1 <= m, n <= 50`
- `1 <= accounts[i][j] <= 100`

```cpp
// https://leetcode.com/problems/richest-customer-wealth/description/
// LeetCode: Richest Customer Wealth

/*
  initialize maxWealth =0
  For every person, add his wealth from all the banks
  ie. for every row,sum of all the column values and store in totalWealth.
  If totalWealth is more than maxWealth, then update maxWealth.
*/

  class Solution {
  public:
      int maximumWealth(vector<vector<int>>& accounts) {
          int maxWealth = 0;
          for(int i =0;i<accounts.size();i++){
              int totalWealth = 0;
              for(int j = 0;j<accounts[i].size();j++){
                  totalWealth+=accounts[i][j];
              }
              if(totalWealth>maxWealth){
                  maxWealth=totalWealth;
              }

          }
          return maxWealth;
      }
  };
```

Sums of i-th row and i-th column

Basic Accuracy: 33.96% Submissions: 13K+ Points: 1

Given a matrix A of dimensions NxM. Check whether the sum of the i^{th} row is equal to the sum of the i^{th} column.

Note: Check only up to valid row and column numbers i.e if the dimensions are 3x5, check only for the first 3 rows and columns, i.e. min(N, M).

Example 1:

Input:
N=2,M=2
A=[[1,2],[2,1]]
Output:
1
Explanation:
The sum of 1st row is equal to sum of 1st column and also sum of 2nd row is equal to the sum of 2nd column.So, Answer is 1.

Example 2:

Input:
N=1,M=3
A=[[5],[0],[0]]
Output:
1
Explanation:
The sum of 1st column is equal to the sum of 1st row.Thus,answer is 1.
(We do not check for the 2nd and 3rd rows because there are no 2nd and 3rd columns.)

Your Task:

You don't need to read input or print anything. Your task is to complete the function **sumOfRowCol()** which takes two integers N, M and a 2D array A as input parameters and returns 1 if all the valid sum of rows is equal to the valid sum of columns. Otherwise, returns 0.

Expected Time Complexity:O(N*M)

Expected Auxillary Space:O(min(N,M))

Constraints:

$1<=N,M,A[i][j]<=10^3$

```
//https://www.geeksforgeeks
    .org/problems/sums-of-i-th-row-and-i-th-column3054/1
// GFG: Sums of i-th row and i-th column

/*
  first calculate number of valid rows and colum u can traverse
  calculate the sum of ith row and ith column and check they are equal or
      not,if they are not equal just return 0
  if for all pairs, the sum of rows and colm are equal then return 1
*/
```

```cpp
class Solution {
  public:
    int sumOfRowCol(int N, int M, vector<vector<int>> A) {
        // caluclate number of valid rows and column
        int valid = min(N, M);

        for (int i = 0; i < valid; ++i) {
            // Calculate the sum of elements in the ith row
            int rowSum = 0;
            for (int j = 0; j < M; ++j) {
                rowSum += A[i][j];
            }

            // Calculate the sum of elements in the ith column
            int colSum = 0;
            for (int j = 0; j < N; ++j) {
                colSum += A[j][i];
            }

            // check wether the sum of ith row and sum of ith colm is equal or not
            if (rowSum != colSum) {
                return 0;  // Return -1 if the condition is not satisfied
            }
        }

        // If all pairs of sums are equal, return 1
        return 1;
    }
};
```

Row with max 1s

Medium Accuracy: 33.09% Submissions: 231K+ Points: 4

Given a boolean 2D array of n x m dimensions, **consisting of only 1's and 0's**, where each row is sorted. Find the 0-based index of the first row that has the maximum number of **1's**.

Example 1:

Input:
N = 4 , M = 4
Arr[][] = {{0, 1, 1, 1},
{0, 0, 1, 1},
{1, 1, 1, 1},
{0, 0, 0, 0}}
Output: 2
Explanation: Row 2 contains **4** 1's (0-based indexing).

Example 2:

Input:
N = 2, M = 2
Arr[][] = {{0, 0}, {1, 1}}
Output: 1
Explanation: Row 1 contains **2** 1's (0-based indexing).

Your Task:
You don't need to read input or print anything. Your task is to complete the function **rowWithMax1s()** which takes the array of booleans **arr[][], n** and **m** as input parameters and returns the 0-based index of the first row that has the most number of 1s. If no such row exists, return -1.

Expected Time Complexity: O(N+M)
Expected Auxiliary Space: O(1)

Constraints:
$1 \leq N, M \leq 10^3$
$0 \leq Arr[i][j] \leq 1$

```cpp
// https://www.geeksforgeeks.org/problems/row-with-max-1s0023/1
// GFG: Row with max 1s

/*
   Initialize two variables, max to track the maximum count of 1s,
   and index to store the corresponding row index.
   in starting we assume that there are now row exits that conatins 1 so
       index=-1,max=0,
   Use a loop to iterate through each row of the matrix.
   For each row, initialize a count variable to keep track of the number of
       1s in that row.
   Compare the count of 1s in the current row with the maximum count so far.
   If the current row has more 1s, update the maximum count (max) and the
       corresponding row index (index).
*/
 class Solution{
 public:
     int rowWithMax1s(vector<vector<int>> arr, int n, int m) {
        int max=0;
        int index=-1;;
        for (int i = 0; i < n; i++) {
            int count=0;
            for (int j = 0; j < m; j++) {
                if(arr[i][j]==1){
                    count++;
                }
            }

            if(count > max){
                max=count;
                index=i;
            }
         }

         return index;
     }

 };
```

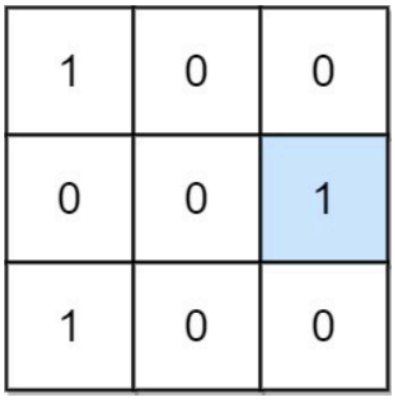
1582. Special Positions in a Binary Matrix

Easy | Topics | Companies | Hint

Given an `m x n` binary matrix `mat`, return *the number of special positions in* `mat`.

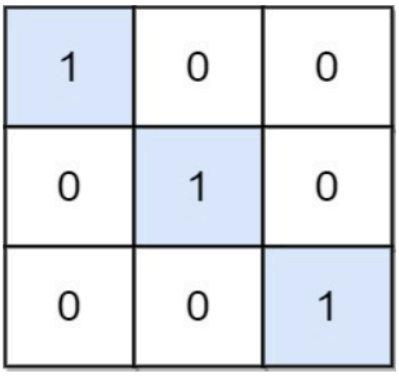
A position `(i, j)` is called **special** if `mat[i][j] == 1` and all other elements in row `i` and column `j` are `0` (rows and columns are **0-indexed**).

Example 1:




```
Input: mat = [[1,0,0],[0,0,1],[1,0,0]]
Output: 1
Explanation: (1, 2) is a special position because mat[1][2] == 1 and all
other elements in row 1 and column 2 are 0.
```

Example 2:




```
Input: mat = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3
Explanation: (0, 0), (1, 1) and (2, 2) are special positions.
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= m, n <= 100`
- `mat[i][j]` is either `0` or `1`.

```
//https://leetcode
    .com/problems/special-positions-in-a-binary-matrix/description/
// LeetCode: Special Positions in a Binary Matrix

/*
Approach->
Counting occurrences:
       You initialize two array, row and col, to store the counts for each
       element in its respective row and column.

Identifying special elements:
        You iterate through the matrix again, checking if an element is
            present
        and if its corresponding row and column counts are both equal to 1.

Counting special elements:
        If both conditions are met, the element is considered "special"
        and you increment  ans variable.

Final result:
       The function returns the final ans of special elements in the matrix.
*/
```

```cpp
class Solution {
public:

    int numSpecial(vector<vector<int>>& mat) {
        // Get the number of rows and columns in the matrix
        int n = mat.size();
        int m = mat[0].size();

        // Arrays to store the count of 1s in each row and each column
        int row[n];
        int col[m];

        // Initialize the count arrays to zero
        for(int i = 0; i < n; i++)
            row[i] = 0;
        for(int j = 0; j < m; j++)
            col[j] = 0;

        // Count the number of 1s in each row and each column
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (mat[i][j] == 1) {
                    row[i]++;
                    col[j]++;
                }
            }
        }
// Count the number of special positions where there is only one 1 in both its row and column
         int ans = 0;
         for (int i = 0; i < n; i++) {
             for (int j = 0; j < m; j++) {
                 if (mat[i][j] == 1 && row[i] == 1 && col[j] == 1) {
                     ans++;
                 }
             }
         }

         // Return the total count of special positions
         return ans;
    }
};
```

66. Plus One

Easy | Topics | Companies

You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the `i^th` digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading `0`'s.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

Example 2:

```
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives 4321 + 1 = 4322.
Thus, the result should be [4,3,2,2].
```

Example 3:

```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

Constraints:

- `1 <= digits.length <= 100`
- `0 <= digits[i] <= 9`
- `digits` does not contain any leading `0`'s.

```cpp
// https://leetcode.com/problems/plus-one/description/
// LeetCode: Plus One

/*
Start traversing through end.
If last element is not 9, just add 1 to it and return the vector.
If last element is 9, make it zero, move ahead (reverse order).
Now if this element is not 9, add 1 to it and return vector, otherwise if it is also 9, make it zero and repeat the 3rd and 4th steps.
Now, if the number is 999, then we have to make it 1000 (999 + 1 = 1000). So, push 0 to the last of our vector (0,0,0) so that it becomes (0,0,0,0) and make first element '1' => 1000.
This is how we add 1 to the number!
*/

class Solution {
public:

    vector<int> plusOne(vector<int>& digits) {
        int n = digits.size();

        for(int i=n-1; i>=0; i--){
            if(digits[i]<9){
                digits[i]++;
                return digits;
            }else{
                digits[i] = 0;
            }
        }

        // to handle '999' => 999 + 1 = 1000
        digits.push_back(0);
        digits[0] = 1;
        return digits;
    }
};
```