1267. Count Servers that Communicate

Depth-First Search Breadth-First Search Union Find

Problem Description

can either contain a server (represented by the number 1) or not contain a server (represented by the number 0). The crucial point to understand is that servers are considered to be able to communicate with each other if and only if they are located in the same row or the same column. The objective is to calculate how many servers are able to communicate with at least one other server within the grid. Intuition

In this problem, we're given a representation of a server center as a m * n integer matrix called grid, where each cell in the grid

Array

Counting

Matrix

To determine the number of servers that can communicate with others, we need a way to check each server and see if there is at

Medium

least one other server on the same row or column. The solution approach can start by keeping two separate arrays, row and col, to keep track of how many servers are on each row and column respectively. We then iterate over the entire grid, counting how many servers are in each row and in each column. This is done by going

through each cell of the matrix, and whenever we encounter a server (a cell with a value of 1), we increment the corresponding

row and column counters by 1.

Once we have the counts of servers in each row and column, we can again iterate over the grid. This time, for each server, we check whether the corresponding row or column has more than one server (which means the count is greater than 1). If so, the server can communicate with others, and it should be included in the final count. We do this check using a generator expression

and pass it to sum function to get the total count of servers that can communicate.

values to 0. These lists hold counts of servers in each row and column.

This approach efficiently factors in both row-wise and column-wise communication, ensuring no double counting of servers. **Solution Approach**

algorithm selectively counts and then aggregates server connections based on the communication criteria. The solution leverages simple data structures – two lists named row and col – to store the counts of servers in each row and column,

The implementation of the solution uses a two-pass algorithm across the rows and columns of the server grid matrix. The

Here's a step-by-step walk-through of the algorithm:

row = [0] * m

col = [0] * n

respectively.

Iterate over each cell in the grid matrix to fill in the row and col arrays. Whenever a server (1 in grid) is found, increment the respective row's and column's count.

Initialize two lists, row and col, with a size equal to the number of rows m and columns n in input grid, respectively, and set all

```
for i in range(m):
   for j in range(n):
```

if grid[i][j]:

row[i] += 1

col[j] += 1

After populating the row and col arrays, we carry out a second iteration over the grid. For each server found, check if it can

```
communicate (i.e., if row[i] or col[j] is greater than 1).
The expression grid[i][j] and (row[i] > 1 \text{ or } col[j] > 1) evaluates to True if there is a server in grid[i][j] and it can
communicate with at least one server in the same row or column.
The generator expression inside the sum function goes through all cells in the grid, evaluates the above condition, and if it is
```

for j in range(n)

during the second iteration.

After initialization:

row = [2, 2, 1]

col = [1, 1, 2, 1]

communicate.

Solution Implementation

rows = [0] * num_rows

columns = [0] * num cols

Python

class Solution:

[1, 0, 0, 1],

True, it contributes 1 toward the sum, effectively counting the server. return sum(grid[i][j] and (row[i] > 1 or col[j] > 1)for i in range(m)

This code bypasses a separate conditional branching for each server and instead uses a compact generator expression to do the counting, which is a common Pythonic pattern for concise and readable code. The main algorithmic insight is that by keeping

track of the row and column server counts, we can determine the capability of each server to communicate in constant time

Initialize two lists, row and col, each with values initialized at 0. For our grid, row will have 3 elements and col will have 4

The row list tells us that the first and second rows each have 2 servers, while the third row has 1 server. The col list tells us

Next, we perform a second iteration over the grid. This time, for each server, we check if it can communicate with others. If

```
Example Walkthrough
  Let's assume we have a server grid grid represented by the following 3 \times 4 matrix:
```

elements, corresponding to the number of rows and columns in grid, respectively.

After completing this step for our example, the row and col lists will look like this:

Let's walk through the solution step-by-step for this example:

col = [0, 0, 0, 0]We iterate over each cell in the grid to count the number of servers in each row and column.

that the first, second, and fourth columns each have 1 server, and the third column has 2 servers.

 \circ For grid[1][1], we have a server and both row[1] > 1 and col[1] > 1, which means this server can communicate.

• For grid[2][2], we do have a server but since row[2] is not greater than 1, this server cannot communicate.

We continue performing this check for every cell in the grid that contains a server (1).

```
    For grid[0][0], we have a server and row[0] > 1 (because row[0] is 2) - so this server can communicate.

    For grid[0][3], we have a server, but neither row[0] > 1 nor col[3] > 1 (because row[0] is 2, but col[3] is 1) - so this server cannot
```

For grid[1][2], we have a server and col[2] > 1, so this server can communicate.

Arrays to keep track of the count of servers in each row and column

Second pass: count the number of servers that can communicate

int numRows = grid.length; // Number of rows in the grid

int countServers(std::vector<std::vector<int>>& grid) {

std::vector<int> serversInRow(rowCount, 0);

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

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

serverCount++;

// it is considered connected.

connectedServers++;

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

std::vector<int> serversInColumn(colCount, 0);

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

// Calculate the number of servers in each row and column

int rowCount = grid.size(); // number of rows in the grid

int serverCount = 0; // variable to keep track of the total count

int colCount = grid[0].size(); // number of columns in the grid

// Create a vector to store the count of servers in each row and column

if (grid[i][j]) { // if there is a server at position (i, j)

++serversInRow[i]; // increment the count for the row

// Count the number of servers that can communicate with at least one other server

if (grid[i][j] && (serversInRow[i] > 1 || serversInColumn[j] > 1)) {

return serverCount; // Return the total count of servers that can communicate

// A server at (i, j) can communicate if there are other servers in the same

// row or column (hence, count will be more than 1 for that row or column).

++serversInColumn[j]; // increment the count for the column

int numCols = grid[0].length; // Number of columns in the grid

if grid[i][j] == 1 and (rows[i] > 1 or columns[j] > 1):

Servers can communicate if they are in the same row or column with another server

Check if there's a server and if it's not the only server in its row or column

row[i] or col[j] is greater than 1, the server can communicate.

def count_servers(self, grid: List[List[int]]) -> int:

num_rows, num_cols = len(grid), len(grid[0])

Number of rows and columns in the grid

rows[i] += 1

for j in range(num_cols):

public int countServers(int[][] grid) {

columns[j] += 1

server_count += 1

Using the generator expression provided, we count the servers that can communicate, which evaluates to 3 for our example (grid[0][0], grid[1][1], and grid[1][2]). Therefore, for the grid given in our example, the total number of servers that can communicate with at least one other server is 3.

First pass: count the number of servers in each row and column for i in range(num_rows): for j in range(num_cols): if grid[i][j] == 1:

```
Java
```

class Solution {

#include <vector>

class Solution {

public:

server_count = 0

return server_count

for i in range(num_rows):

```
// Arrays to store the count of servers in each row and column
        int[] rowCount = new int[numRows];
        int[] colCount = new int[numCols];
       // First iteration to fill in the rowCount and colCount arrays
        for (int i = 0; i < numRows; ++i) {
            for (int j = 0; j < numCols; ++j) {</pre>
                // Count servers in each row and column
                if (grid[i][j] == 1) {
                    rowCount[i]++;
                    colCount[j]++;
        // Counter for the total number of connected servers
        int connectedServers = 0;
       // Second iteration to count the servers that can communicate
        // Servers can communicate if they are not the only one in their row or column
        for (int i = 0; i < numRows; ++i) {</pre>
            for (int j = 0; j < numCols; ++j) {</pre>
                if (grid[i][j] == 1 && (rowCount[i] > 1 || colCount[j] > 1)) {
                    connectedServers++;
        // Return the number of connected servers
        return connectedServers;
C++
```

```
};
TypeScript
function countServers(grid: number[][]): number {
    // Get the number of rows (m) and columns (n) from the grid.
    const rowCount = grid.length;
    const colCount = grid[0].length;
    // Initialize arrays to keep track of server counts in each row and column.
    const rowServerCounts = new Array(rowCount).fill(0);
    const colServerCounts = new Array(colCount).fill(0);
    // First pass: Count the number of servers in each row and column.
    for (let rowIndex = 0; rowIndex < rowCount; rowIndex++) {</pre>
        for (let colIndex = 0; colIndex < colCount; colIndex++) {</pre>
            if (grid[rowIndex][colIndex] === 1) {
                rowServerCounts[rowIndex]++;
                colServerCounts[colIndex]++;
    // Initialize a variable to keep track of the total number of connected servers.
    let connectedServers = 0;
    // Second pass: Determine if each server is connected horizontally or vertically.
    for (let rowIndex = 0; rowIndex < rowCount; rowIndex++) {</pre>
        for (let colIndex = 0; colIndex < colCount; colIndex++) {</pre>
```

```
// Return the total count of connected servers.
      return connectedServers;
class Solution:
   def count_servers(self, grid: List[List[int]]) -> int:
       # Number of rows and columns in the grid
       num_rows, num_cols = len(grid), len(grid[0])
       # Arrays to keep track of the count of servers in each row and column
       rows = [0] * num_rows
       columns = [0] * num_cols
       # First pass: count the number of servers in each row and column
       for i in range(num_rows):
           for j in range(num_cols):
               if grid[i][j] == 1:
                    rows[i] += 1
                   columns[j] += 1
       # Second pass: count the number of servers that can communicate
       # Servers can communicate if they are in the same row or column with another server
       server_count = 0
       for i in range(num_rows):
           for j in range(num_cols):
               # Check if there's a server and if it's not the only server in its row or column
               if grid[i][j] == 1 and (rows[i] > 1 or columns[j] > 1):
                   server_count += 1
       return server_count
Time and Space Complexity
```

// If there's a server and there's more than one server in the current row or column,

if (grid[rowIndex][colIndex] === 1 && (rowServerCounts[rowIndex] > 1 || colServerCounts[colIndex] > 1)) {

The given code consists of two main parts: The first part is a double loop that goes through the entire grid to count the number of servers in each row and column. Since this loop goes through all the elements of the m x n grid exactly once, the time

Time Complexity

complexity for this part is 0(m * n). The second part of the code calculates the sum with a generator expression that also iterates through every element in the grid.

It checks if there's a server in the given cell (grid[i][j]) and if there is more than one server in the corresponding row or column.

Since this is done for each element, it also has a time complexity of 0(m * n). Therefore, the total time complexity of the entire function is 0(m * n) + 0(m * n), which simplifies to 0(m * n) because we only take the highest order term for big O notation.

Space Complexity

For space complexity, the code uses additional arrays row and col to store the counts of servers in each row and column, respectively. The size of row is m and the size of col is n. Hence, the extra space used is 0(m + n).

In conclusion, the time complexity is 0(m * n) and the space complexity is 0(m + n).