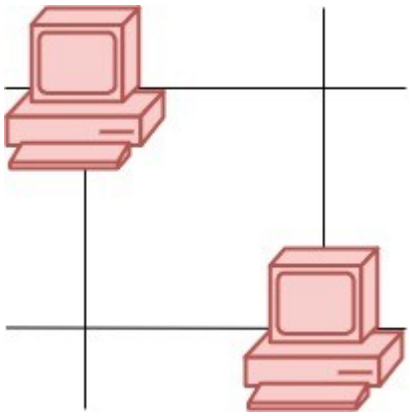
1267. Count Servers that Communicate

You are given a map of a server center, represented as a `m * n` integer matrix `grid`, where 1 means that on that cell there is a server and 0 means that it is no server. Two servers are said to communicate if they are on the same row or on the same column.

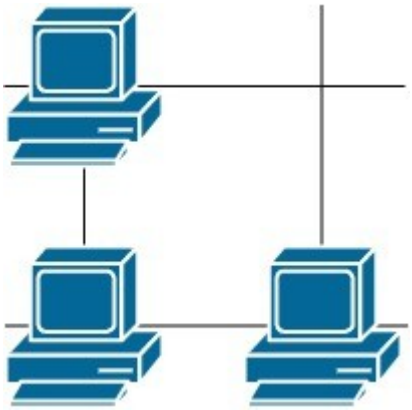
Return the number of servers that communicate with any other server.

Example 1:



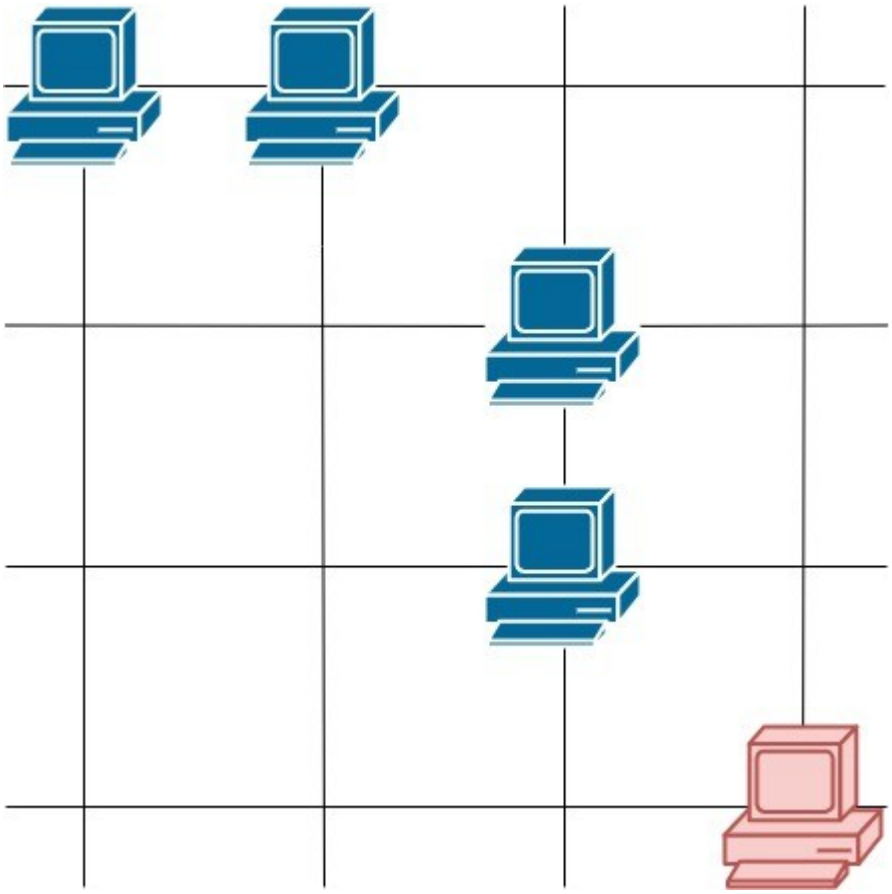
Input: grid = [[1,0],[0,1]]
Output: 0
Explanation: No servers can communicate with others.

Example 2:



Input: grid = [[1,0],[1,1]]
Output: 3
Explanation: All three servers can communicate with at least one other server.

Example 3:



```
Input: grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]]
Output: 4
Explanation: The two servers in the first row can communicate with each other. The two servers in the third column can communicate with each other. The server at right bottom corner can't communicate with any other server.
```

Overview

We are given a grid representing a server center in the form of a matrix of size `m x n`. Each cell of the matrix contains either a `1`, indicating the presence of a server, or a `0`, indicating an empty space.

We need to return the number of servers that can communicate with at least one other server. This excludes servers that are isolated, i.e., those that do not share a row or column with any other server.

The first thing to note is that a server can communicate with another server if they are located either in the same row or the same column. Thus, the key observation here is that we only need to check rows and columns to determine if a server is communicable. If there’s at least one other server in the same row or column, then this server is communicable.

1267. Count Servers that Communicate

```
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;

class DSU{
    public:
        vi parent;
        vi group;
    public:
        DSU(int n){
            group.resize(n,1);
            parent.resize(n);
            std::iota(parent.begin(),parent.end(),0);
        }

        int find(int p){
            if(p==parent[p]) return p;
            return parent[p]=find(parent[p]);
        }

        int unify(int p,int q){
            int parent_p=find(p);
            int parent_q=find(q);

            if(parent_p==parent_q) return false;

            if(group[parent_p]<group[parent_q]){
                group[parent_q]+=group[parent_p];
                group[parent_p]=1;
                parent[parent_p]=parent_q;
            }
            else{
                group[parent_p]+=group[parent_q];
                group[parent_q]=1;
                parent[parent_q]=parent_p;
            }
            return true;
        }
};
```

```
class Solution {
    public:
        int countServers(vvi& grid){
            int m=grid.size();
            int n=grid[0].size();

            DSU dsu=DSU(m+n);

            // Iterate over the grid and group all servers in same row and column
            int cnt_unify=0,cnt_already_unified=0;
            for(int row=0;row<m;++row){
                for(int col=0;col<n;++col){
                    if(grid[row][col]==1){
                        if(dsu.unify(row,col+m)) cnt_unify++;
                        else cnt_already_unified++;
                    }
                }
            }

            // Count isolate servers
            int isolate_servers=0;
            for(auto&g: dsu.group) isolate_servers+=(g==2);

            return cnt_unify+cnt_already_unified-isolate_servers;

        }
};
```