1254. Number of Closed Islands Medium

Before starting to solve this problem I would recommend you solving Number of Islands if you haven't solved it.

Let's start by understanding the problem statement, Given a 2D matrix named grid with only 0's and 1's. 0 represents land whereas 1 represents water. All we need to find the number of closed islands.

First of all, what is Island? An Island is a piece of land surrounded by water. Here it is a maximal 4-directionally connected group of 0's. Then what makes them closed Island? If an Island is surrounded by water i,e. 1's in all four directions, then it is a closed Island.

NOTE: Generally in these types of questions, 1's will be considered as land and 0's as water. But here it's vice versa.

Intuition:

Now we know, a closed island is a group of connected 0's surrounded by 1's in all four sides. The 0's in the edges of the matrix doesn't make a closed island, since it's only possible for it to be surrounded by 1's in any 3 sides or less.

The idea is to count the number of islands(0's) surrounded by water(1's) using either DFS/BFS traversal.

Approach: DFS Traversal

Algorithm:

1)Create a function to perform the DFS traversal.

2)Pass the grid(Matrix), index of the element in the grid as a parameter to the DFS function.If the current element is 0, mark it as visited in the same matrix. The function executes recursively for all the adjacent indexes till the value in the respective index is 0. i.e,

```
if(grid[i][j]==0){
   grid[i][j]=1;//mark it as visited
  //function call for adjacent indexes.
}
```

Once the element in the matrix is 1 or the index exceeds the boundary of the grid then the recursive call of the DFS function is returned.

i.e, if (i>=row or j>=column or i<0 or j<0 or grid[i][j]==1){ return;}

3) Perform DFS traversal on all edges of the given matrix if the value of the matrix in edges is 0.

4)Iterate over the matrix, whenever an unvisited 0 is encountered perform DFS traversal and count the number of closed islands. The count of closed islands is the number of times dfs function called for the remaining unvisited cells.

Ie, iterate matrix

```
if (grid[i][j]==0){
   //call dfs function
  count+=1;
}
```

Code: Python

```python
class Solution:
    def dfs(i,j,grid,r,c):
        if i>=r or j>=c or i<0 or j<0 or grid[i][j]==1:
            return
        else:
            grid[i][j]=1
            Solution.dfs(i+1,j,grid,r,c)
            Solution.dfs(i-1,j,grid,r,c)
            Solution.dfs(i,j+1,grid,r,c)
            Solution.dfs(i,j-1,grid,r,c)

    def closedIsland(self, grid: List[List[int]]) -> int:
        r=len(grid)
        ans=0
        if r!=0:
            c=len(grid[0])

        for i in range(r):
            if(grid[i][0]==0):
                Solution.dfs(i,0,grid,r,c)
            if (grid[i][c-1]==0):
                Solution.dfs(i,c-1,grid,r,c)

        for i in range(c):
            if(grid[0][i]==0):
                Solution.dfs(0,i,grid,r,c)
            if (grid[r-1][i]==0):
                Solution.dfs(r-1,i,grid,r,c)

        for i in range(r):
            for j in range(c):
                if (grid[i][j]==0):
                    ans+=1
                    Solution.dfs(i,j,grid,r,c)
        return ans
```

C++

```
void dfs1(int n,int m,vector<vector<int>>& grid,int &r,int &c){
    if((n>=r) || (m>=c) || (m<0) || (n<0) || (grid[n][m]==1)){
        return;
    }
    else{
        grid[n][m]=1;
        dfs1(n+1,m,grid,r,c);
        dfs1(n-1,m,grid,r,c);
        dfs1(n,m+1,grid,r,c);
        dfs1(n,m-1,grid,r,c);
    }
}
class Solution {
public:
    int closedIsland(vector<vector<int>>& grid) {
        int r=grid.size();
        if(r==0){
            return 0;
        }
        int c=grid[0].size();
        for(int i=0;i<r;i++){
            if(grid[i][0]==0){
                dfs1(i,0,grid,r,c);
            }
            if(grid[i][c-1]==0){
                dfs1(i,c-1,grid,r,c);
            }
        }
        for(int i=0;i<c;i++){
            if(grid[0][i]==0){
                dfs1(0,i,grid,r,c);
            }
            if(grid[r-1][i]==0){
                dfs1(r-1,i,grid,r,c);
            }
        }
        int ans=0;
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                if(grid[i][j]==0){
                    dfs1(i,j,grid,r,c);
                        ans++;
                }
            }
        }
        return ans;
    }
};
```

Time Complexity: O(rxc)

Space Complexity: O(1)