**Number of Islands:**

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

**Input:** grid = [ ["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

**Output:** 1

**Intuition**

Intuition for this problem is very simple. We just need to traverse this grid starting from land(1) until we find water(0) and finally return the count.

**Approach**

This problem can be solved by DFS or BFS technique, but I feel that DFS is more appropriate here.

Think about the boundary case.

Start traversing grid[i][j]=='1' and mark them anything apart from '0' and '1' to denote it is already visited.

Complexity

Time complexity:

O(N)

Space complexity:

O(1)

classSolution{

public:

voiddfs(inti, intj, introw,intcol,vector<vector<char>>&grid)

{

if(i<0||i>=row||j<0||j>=col||grid[i][j]!='1') return;

grid[i][j]='2';

dfs(i+1,j,row,col,grid);

dfs(i-1,j,row,col,grid);

dfs(i,j+1,row,col,grid);

dfs(i,j-1,row,col,grid);

}

intnumIslands(vector<vector<char>>&grid) {

introw = grid.size();

if(row==0) return0;

intcol = grid[0].size();

intcount = 0;

for(inti=0;i<row;i++)

{

for(intj=0;j<col;j++)

{

if(grid[i][j]=='1')

{

dfs(i,j,row,col,grid);

count +=1;

}

}

}

returncount;

}

};

**Number of Provinces**

There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an n\*n matrix isConnected where isConnected[i][j]=1 if the ith city and the jth city are directly connected, and isConnected[i][j]=0 otherwise.

Return *the total number of* ***provinces***.

classSolution{

public:

intfindCircleNum(vector<vector<int>>&isConnected) {

intn = isConnected.size();

vector<int> adj[n];

for(inti=0;i<n;i++)

{

for(intj=0;j<n;j++)

{

if(isConnected[i][j]==1&& i!=j)

{

adj[i].push\_back(j);

}

}

}

intcount = 0;

vector<int> vis(n,0);

for(inti=0;i<n;i++){

if(!vis[i])

{

count++;

dfs(i,adj,vis);

}

}

returncount;

}

voiddfs(intindex, vector<int> adj[],vector<int>&vis)

{

vis[index] = 1;

for(autoi:adj[index])

{

if(!vis[i])

{

dfs(i,adj,vis);

}

}

}

};