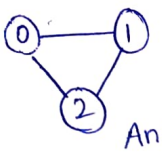
Graph Questions

(1) Find if Path exists in graph



Answer: True

Given, n = 3
edges = [[0,1],[1,2],[0,1]]
src = 0
dest = 1.

Approach:

Case I: There will be a path b/w src and dest only and only if they are in the same component.

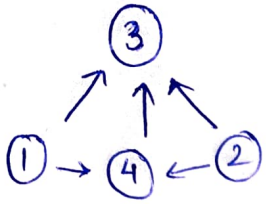
Case II: There will be no path if src and dest are not in same component.

So, we can use any traversal (BFS or DFS) and ~~the~~ • if nodes in between comes out to be destinat^n. It shows they are in the same component. Hence there is a path b/w src and dest. Return True. otherwise False, if dest nodes is not found while traversing.

Code:

```
bool isValidPath(int n, vector<vector<int>>&edges, int ~~src~~ src, int dest){
    vector<int> vis(n,0);
    if(src == dest)
        return true;                          → Corner case
    ~~unorderd~~ unordered_map<int, vector<int>> adj;
    for(auto &it : edges){
        adj[it[0]].push_back(it[1]);          → convert adjacency
        adj[it[1]].push_back(it[0]);            matrix to adjacency
    }                                           list by using
                                                unordered map.
    queue<int> q;
    q.push(src);
    vis[src]=1;
    while(!q.empty){
        int node = q.front();                 → simple BFS traversal
        q.pop();                                with simple condition
        for(auto &it : adj[node]){
            if(it == dest)                    → It returns if there is
                return true;                    a path ~~or not~~
            if(vis[it]==0){
                vis[it]=1;
                q.push(it);
            }
        }
    }
    return false;
}
```

(2) Find the Town Judge



Given, n = 4

trust = [[1, 3], [4, 3], [2, 3], [1, 4], [2, 4]]

Output = 3

Approach: To be a Town Judge, There are 2 conditions:

(i) Town Judge doesn't trust any body/node.

(ii) All other nodes trust Town Judge

Intution is that 1 → 3 which denotes 1 trusts 3 and 1 goes to 3, where outdegree of 1 increases by 1 and indegree of 3 increase by 1.

To be a Town Judge,

Indegree = n-1 and

Outdegree = 0.

We can achieve by just traversing the adjacency matrix and maintain 2 vectors indeg and outdeg increase indegree and outdegree as the edges comes out.

Code:

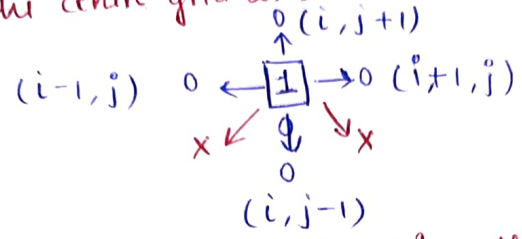
```
int findJudge(int n, vector<vector<int>>& trust){
    vector<int> in(n+1, 0), out(n+1, 0);
    for(auto &it : trust){
    out[it[0]]++;
    in[it[1]]++;
    }
    for(i → n){
        if(in[i] == n-1 && out[i]==0)
            return i;
    }
    return -1;
}
```

(3) Number of Islands

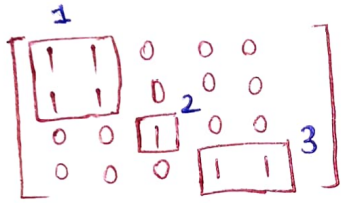
Given, grid = $\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$

Output = 3

Explanation: Area is said to be Island only and only if left, right, up and downward grid is 0, and the centre grid is 1:

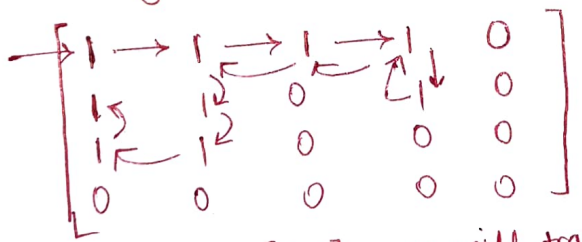


{we are not considering diagnolly}



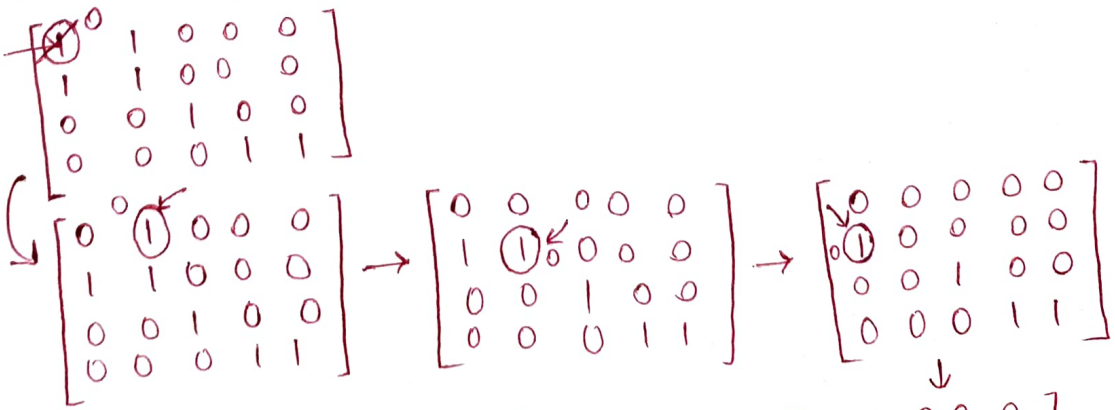
Output = 3

Approach: We have to make the land as water (1→0) after traversing it and after adding it to our island.



Starting from [0,0], we will travere it by DFS and all the connected 1's will be called and the output is 1 here.

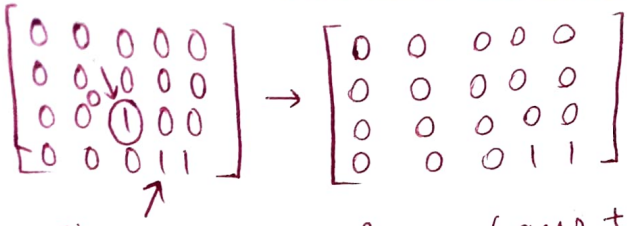
DFS (1) →



Now, count of island will increase by 1. ans = 1

Now,

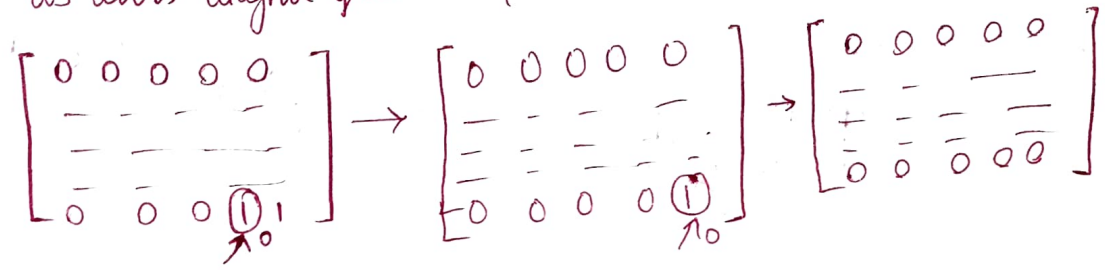
DFS(2) →



This won't be consider as this diagnol grid

{ans++}
{ans = 2}

DFS(3) →



ans++
{ans = 3}

```
bool isValid(int i, j, n, m, grid) {
    if(i >= 0 && i < n && j >= 0 && j < m, grid[i][j] == '1')
        return true;
    return false;
}
void dfs(i, j, n, m, grid) {
    grid[i][j] = '0';
    if(isValid(i, j+1, n, m, grid))
        dfs(i, j+1, n, m, grid);
    if(isValid(i+1, j, n, m, grid))
        dfs(i+1, j, n, m, grid);
    if(isValid(i, j-1, n, m, grid))
        dfs(i, j-1, n, m, grid);
    if(isValid(i-1, j, n, m, grid))
        dfs(i-1, j, n, m, grid);
}
int numberofislands(grid) {
    int n, m;
    int ans;
    for(i→n)
        for(j→m) {
            if(grid[i][j] == '1') {
                ans++;
                dfs(i, j, n, m, grid);
            }
    return ans; }.
```