**547. Number of Provinces :-**

Medium Accepted: 661.3K Submissions: 1M Acceptance Rate: 64.7%

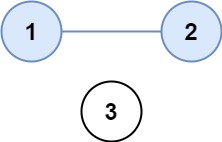
There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

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

You are given an n x 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***.

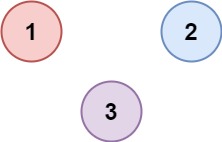
**Example 1:**



**Input:** isConnected = [[1,1,0],[1,1,0],[0,0,1]]

**Output:** 2

**Example 2:**



**Input:** isConnected = [[1,0,0],[0,1,0],[0,0,1]]

**Output:** 3

**Constraints:**

* 1 <= n <= 200
* n == isConnected.length
* n == isConnected[i].length
* isConnected[i][j] is 1 or 0.
* isConnected[i][i] == 1
* isConnected[i][j] == isConnected[j][i]

**Code :-**

class Solution {

public:

    int findCircleNum(vector<vector<int>>& connect) {

        int n = connect.size();

        if(n==1)    return 1;

        vector<bool> dp(n+1, false);

        unordered\_map<int, vector<int>> mp;

        for(int i=0; i<n-1; i++){

            for(int j=i+1; j<n; j++){

                if(connect[i][j]==1){

                    mp[i+1].push\_back(j+1);

                    mp[j+1].push\_back(i+1);

                }

            }

        }

        int component = 0;

        for(int i=1; i<=n; i++){

            queue<int> q;

            q.push(i);

            if(dp[i]==false){

                dp[i]=true;

                while(q.empty()==false){

                    int count = q.size();

                    int front = q.front();

                    q.pop();

                    for(int k=1; k<=count; k++){

                        for(auto item:mp[front]){

                            if(dp[item]==false){

                                q.push(item);

                                dp[item] = true;

                            }

                        }

                    }

                }

                component++;

            }

        }

        return component;

    }

};

**T.C :- O( n+1 )**

**S.C :- O( no. of 1’s – no. of diagonal items in matrix )**