**Making A Large Island: -**

**Hard** Accuracy: **52.36%** Submissions: **14K+** Points: **8**

You are given an**n x n binary matrix**grid. You are allowed to change **at most one 0** to be **1**. A group of **connected 1s** forms an island. Two 1s are connected if they share one of their sides with each other.

Return the size of the **largest island** in the grid after applying this operation.

**Example 1:**

**Input:**  
grid = [[1,0],[0,1]]  
**Output:**  
3  
**Explanation:**  
Change any one 0 to 1 and connect two 1s, then we get an island with area = 3.

**Example 2:**

**Input:**  
grid = [[1,1],[1,0]]  
**Output:**  
4  
**Explanation:**  
Change the only 0 to 1 and make the island bigger, then we get an island with area = 4.

**Example 3:**

**Input:**  
grid = [[1,1],[1,1]]  
**Output:**  
4  
**Explanation:**  
Can't change any 0 to 1, only one island possible with area = 4.

**Your Task:**

You don't have to read input or print anything. You have to complete the functions **largestIsland()**, which takes binary matrix as input and returns an integer denoting the **size of the largest island**in grid after applying the operation.

**Constraints:**

1 <= n <= 500  
0 <= grid[i][j] <= 1 (grid[i][j] ∈ {0, 1} and 0<=i,j<n)

**Code: -**

//{ Driver Code Starts

#include<bits/stdc++.h>

using namespace std;

// } Driver Code Ends

//User function Template for C++

int n;

int dx[] = {0, -1, 0, 1};

int dy[] = {-1, 0, 1, 0};

class Solution

{

public:

int helper(vector<vector<int>>& grid, int row, int col, vector<vector<bool>>& vis, int &islandindex){

int ans = 0, fx, fy, nx, ny;

queue<pair<int,int>> q;

q.push({row, col});

while(q.size()){

int count = q.size();

while(count--){

fx = q.front().first;

fy = q.front().second;

q.pop();

grid[fx][fy] = islandindex;

ans++;

for(int i = 0; i<4; ++i){

nx = fx + dx[i];

ny = fy + dy[i];

if(0<=nx and nx<n and 0<=ny and ny<n and !vis[nx][ny] and grid[nx][ny]==1){

vis[nx][ny] = true;

q.push({nx, ny});

}

}

}

}

return ans;

}

int largestIsland(vector<vector<int>>& grid){

n = grid.size();

int islandindex = 10;

vector<vector<bool>> vis(n, vector<bool>(n, false));

unordered\_map<int,int> index;

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

for(int j = 0; j<n; ++j){

if(!vis[i][j] and grid[i][j] == 1){

vis[i][j] = true;

int islandsize = helper(grid, i, j, vis, islandindex);

index[islandindex] = islandsize;

++islandindex;

}

}

}

int ans = INT\_MIN;

int nx, ny;

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

for(int j = 0; j<n; ++j){

// 0 found and have to change

if(grid[i][j] == 0){

unordered\_map<int,bool> taken;

int maxislandpossible = 1;

// go for every direction and pick unique index islands

for(int k = 0; k<4; ++k){

nx = i + dx[k];

ny = j + dy[k];

if(0<=nx and nx<n and 0<=ny and ny<n and grid[nx][ny]>=10){

int curindex = grid[nx][ny];

if(taken[curindex] == false){

taken[curindex] = true;

maxislandpossible += index[curindex];

}

}

}

ans = max(ans, maxislandpossible);

}

}

}

return (ans != INT\_MIN) ? ans : n\*n;

}

};

//{ Driver Code Starts.

int main(){

int t;

cin >> t;

while(t--){

int n;

cin>>n;

vector<vector<int>>grid(n,vector<int>(n));

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

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

cin>>grid[i][j];

Solution ob;

cout<<ob.largestIsland(grid)<<endl;

}

return 0;

}

// } Driver Code Ends

**T.C: - O(N2)**

**S.C: - O(N2)**