MICROSOFT PREPARATION

DAY : 10

Unit Area of Largest region of 1's:

Problem Link :
https://practice.geeksforgeeks.org/problems/length-of-largest-region-of-1s-1587115620/1?page=1&company%5B%5D=Microsoft&category%5B%5D=Graph&sortBy=submissions

Test Cases Passed : 97 / 97

Time Used : 04.20

Difficulty Level : MEDIUM

Approach Used :

DFS():
- Calculating the dimensions of the grid
- Marking the node as visited
- Incrementing the size of component by 1
- Traversing for all the adjacent components :
    - Calculating the indexes of the adjacent elements
    - Checking for the validity of the indexes :
        - Checking if the adjacent node is unvisited and is a 1 :
            - Making a dfs call to mark the node and its adjacent elements as dfs(adj_row,adj_col,visited,grid,size)

findMaxArea():
- Calculate the dimensions of the grid
- Creating a visited vector having an initialization of 1 for all elements
- Creating a max_size variable and assigning it 0
- Traversing through all of the components :
    - Checking if a component is not visited and is 1 :
        - Initializing a size variable with 0 to mark size of component
        - Calling a dfs function to mark the component and update its size as dfs(row,col,visited,grid,component_size)
        - Updating the max_size as max_size = max(max_size, component_size)
- Returning the max_size

Solution :

```
void dfs(int row,int
col,vector<vector<int>>&visited,vector<vector<int>>&grid,int &size)
    {
        // calculating the dimensions of the grid
        int n  = grid.size();
        int m = grid[0].size();

        // marking the node as visited and incrementing the size
        visited[row][col] = 1;
        size++;

        // traversing for all adjacent elements in 8 directions
        for(int i=-1;i<=1;i++)
        {
            for(int j=-1;j<=1;j++)
            {
                // calculating row
                int nrow = row+i;
                // calculating column
                int ncol = col+j;

                // checking for validity of row and columns
                if(nrow<n && ncol<m && nrow>=0 && ncol>=0)
                {

                    // checking if the adjacent element is not already
accessed and is a 1
                    if(!visited[nrow][ncol] && grid[nrow][ncol]==1)
                    {
                        // making a dfs call to mark the adjacent element
and its neighboring elements
                        dfs(nrow,ncol,visited,grid,size);
                    }
                }
            }
        }

    }
    int findMaxArea(vector<vector<int>>& grid) {
        // We will perform something like a DFS algorithm to count the
```

```
largest area
        // calculating the dimensions of the grid
        int n = grid.size();
        int m = grid[0].size();
        // create a visited vector to contain which area is all visited
        vector<vector<int>> visited(n,vector<int>(m,0));

        // traversing through all components and initially considering a
variable maxsize as 0
        int maxsize = 0;

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

            for(int j=0;j<m;j++)
            {
                // if not visited its a new component of 1s and we have to
get max of maxsize and component size
                if(!visited[i][j] && grid[i][j]==1)
                {   // initializing a variable that will contain the size
of current component with 0
                    int size = 0;
                    // calling the dfs function to mark the current
component and update the size of the component
                    dfs(i,j,visited,grid,size);
                    // updating the maximum size
                    maxsize  = max(maxsize,size);
                }
            }
        }
        // returning the maximum size
        return maxsize;
    }
```