MICROSOFT PREPARATION

DAY : 16

Covid Spread :

Problem Link :
https://practice.geeksforgeeks.org/problems/269f61832b146dd5e6d89b4ca18cbd2a2654ebbe/1?page=2&company%5B%5D=Microsoft&category%5B%5D=Graph&sortBy=submissions

Test Cases Passed : 82 / 82

Time Used : 12.00

Difficulty Level : MEDIUM

Approach Used :

- Calculating the dimensions of the grid
- Creating a vector visited that is initialized with 0
- Creating a queue containing the time and node
- Traversing for all elements :
    - Inserting the infected element into the queue with time 0
    - Marking the infected node as visited
- Creating a variable to store the max time and initialize it with 0
- Traversing until the queue becomes :
    - Extracting the first element from the queue
    - Updating the maxtime variable as maxtime = max(time, maxtime)
    - Popping the first element from the queue
    - Traversing for the adjacent elements :
        - Checking for the validity of the adjacent elements
            - Checking if adjacent element is 1 and unvisited :
                - Pushing the element into the queue with 1 more time than required as (time+1, adjacent node)
- Checking if any person remains un infected :
    - Return -1 // cause all cannot get infected
- Return max time variable // all get infected

Solution :

```
int helpaterp(vector<vector<int>> grid)
    {
       // we will be using BFS to solve this algorithm
       // calculating the dimensions of the grid
       int n = grid.size();
       int m = grid[0].size();
       // creating a visited vector to mark the elements as visited and
initializing all as 0
       vector<vector<int>> visited(n,vector<int>(m,0));
       // creating a queue to store the row and col and time
       queue<pair<int, pair<int, int>>> q;
       // marking the already covid patients with time 0
       for(int i=0;i<n;i++)
       {
           for(int j=0;j<m;j++)
           {
               if(grid[i][j]==2)
               {
                   // marking elements as visited
                   visited[i][j]=1;
                   // inserting into queue with time 0
                   q.push({0,{i,j}});
               }
           }
       }
       int timemax = 0;
       // traversing until the queue becomes empty
       while(!q.empty())
       {
           // getting the first element of the queue
           auto it = q.front();
           int times = it.first;
           int row = it.second.first;
           int col = it.second.second;
           // removing the first element from the queue
           q.pop();
           timemax = max(timemax,times);
           // traversing the adjacent elements
           int delRow[] = {-1,0,1,0};
           int delCol[] = {0,1,0,-1};
```

```
            for(int i=0;i<4;i++)
            {
                // getting the indexes of the adjacent elements
                int nrow = row+delRow[i];
                int ncol = col+delCol[i];
                // checking for validity
                if(nrow<n && ncol<m && nrow>=0 && ncol>=0)
                {
                    // checking if element is unvisited and is a person
                    if(!visited[nrow][ncol] && grid[nrow][ncol]==1)
                    {
                        // mark it as visited
                        visited[nrow][ncol] =1;
                        // updating time and pushing to queue
                        q.push({times+1,{nrow,ncol}});
                    }
                }
            }
        }
        // checking if a patient remains unaffected
        for (int i = 0; i < n; i++) {
             for (int j = 0; j < m; j++) {
                 if (grid[i][j] == 1 && !visited[i][j]) {
                     return -1;
                 }
             }
         }
         // returning the max time when all will be getting infected
         return timemax;
    }
```