MICROSOFT PREPARATION

DAY : 11

Flood Fill :

Problem Link : https://leetcode.com/problems/flood-fill/

Test Cases Passed : 297 / 297

Time Used : 06.45

Difficulty Level : EASY

Approach Used :

DFS():

- Calculating the dimensions of the grid
- Marking the node as visited
- Change color of the node to new color in result vector
- Traverse for all adjacent components of the node :
    - Check for validity of the dimensions of adjacent node :
        - Check if node is unvisited and is having same color as source color :
            - Make dfs call to mark it and its neighbors as dfs(adjrow,adjcol,visited,image,result,color,sourcecolor)

findMaxArea():

- Calculate the dimensions of the grid
- Creating a visited vector having an initialization of 1 for all elements
- Creating a result vector having same elements as of image
- Store the color of the source into a variable
- Traverse for the elements that can connect to the source using the dfs call as dfs(sourcerow,sourcecol,visited,image,result,color,sourcecolor)
- Return result vector

Solution :

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

        // marking as visited
        visited[row][col] = 1;
        // changing the color
        result[row][col] = color;
        // traversing the adjacent element
        int delRow[] = {-1,0,1,0};
        int delCol[] = {0,1,0,-1};

        // traversing the adjacent elements
        for(int i=0;i<4;i++)
        {
            int nrow = row+delRow[i];
            int ncol = col+delCol[i];

            // checking if its unvisited and valid
            if(nrow<n && ncol<m && nrow>=0 && ncol>=0 && !visited[nrow][ncol] &&
image[nrow][ncol]== sourcecolor)
            {
                dfs(nrow,ncol,visited,image,result,color,sourcecolor);
            }
        }
    }
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int
color) {
        // calculating the dimensions of the grid
        int n = image.size();
        int m = image[0].size();
        // creating a visited vector
        vector<vector<int>> visited(n,vector<int>(m,0));
        // creating a result vector
        vector<vector<int>> result = image;
        // marking the flood fill using the dfs call
        int sourcecolor = image[sr][sc];
        dfs(sr,sc,visited,image,result,color, sourcecolor);
        return result;
    }
};
```