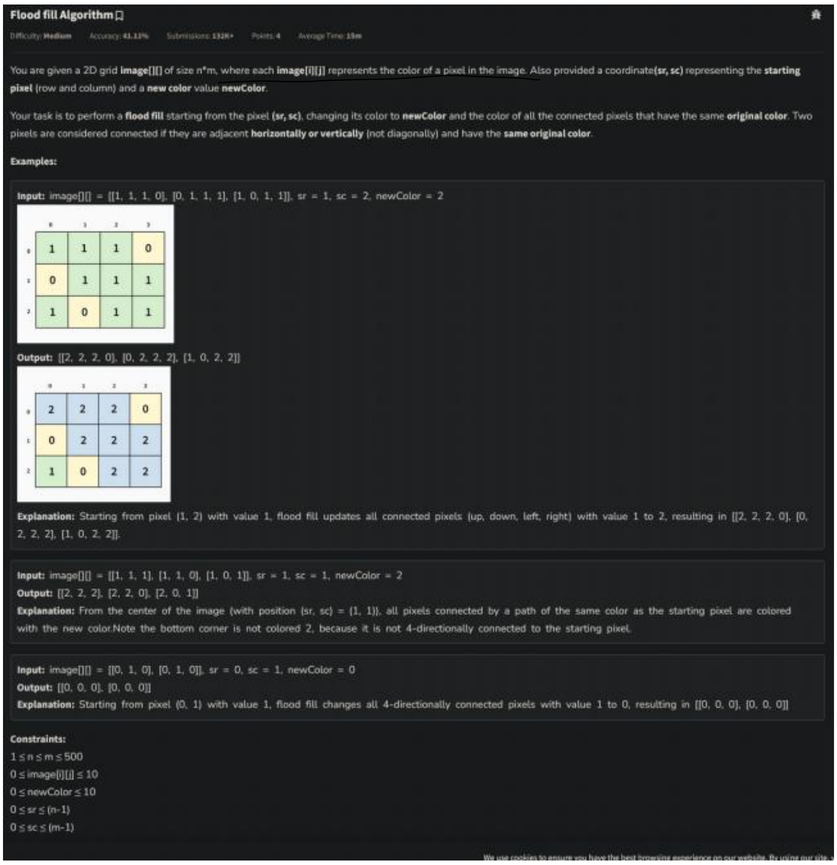
Flood Fill Algorithm :-



Flood fill Algorithm

Difficulty: **Medium** Accuracy: **41.11%** Submissions: **132K+** Points: **4** Average Time: **15m**

You are given a 2D grid **image[][]** of size n*m, where each **image[i][j]** represents the color of a pixel in the image. Also provided a coordinate**(sr, sc)** representing the **starting pixel** (row and column) and a **new color** value **newColor**.

Your task is to perform a **flood fill** starting from the pixel **(sr, sc)**, changing its color to **newColor** and the color of all the connected pixels that have the same **original color**. Two pixels are considered connected if they are adjacent **horizontally or vertically** (not diagonally) and have the **same original color**.

Examples:

Input: image[][] = [[1, 1, 1, 0], [0, 1, 1, 1], [1, 0, 1, 1]], sr = 1, sc = 2, newColor = 2

	0	1	2	3
0	1	1	1	0
1	0	1	1	1
2	1	0	1	1

Output: [[2, 2, 2, 0], [0, 2, 2, 2], [1, 0, 2, 2]]

	0	1	2	3
0	2	2	2	0
1	0	2	2	2
2	1	0	2	2

Explanation: Starting from pixel (1, 2) with value 1, flood fill updates all connected pixels (up, down, left, right) with value 1 to 2, resulting in [[2, 2, 2, 0], [0, 2, 2, 2], [1, 0, 2, 2]].

Input: image[][] = [[1, 1, 1], [1, 1, 0], [1, 0, 1]], sr = 1, sc = 1, newColor = 2
Output: [[2, 2, 2], [2, 2, 0], [2, 0, 1]]
Explanation: From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected by a path of the same color as the starting pixel are colored with the new color.Note the bottom corner is not colored 2, because it is not 4-directionally connected to the starting pixel.

Input: image[][] = [[0, 1, 0], [0, 1, 0]], sr = 0, sc = 1, newColor = 0
Output: [[0, 0, 0], [0, 0, 0]]
Explanation: Starting from pixel (0, 1) with value 1, flood fill changes all 4-directionally connected pixels with value 1 to 0, resulting in [[0, 0, 0], [0, 0, 0]]

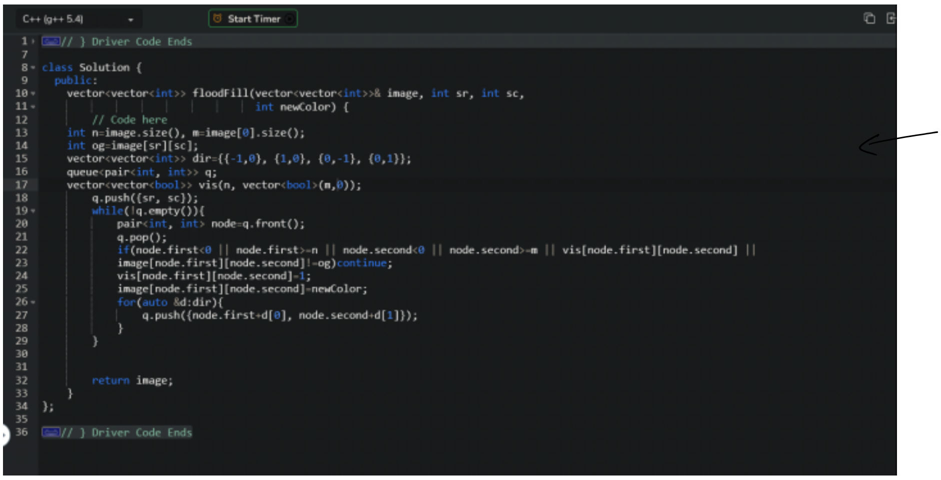
Constraints:
1 ≤ n ≤ m ≤ 500
0 ≤ image[i][j] ≤ 10
0 ≤ newColor ≤ 10
0 ≤ sr ≤ (n-1)
0 ≤ sc ≤ (m-1)

BFS

$$eg: \text{Image}[][] = \begin{matrix} 1, & 1, & 1 \\ 1, & \boxed{1}, & 0 \\ 1, & 0, & 1 \end{matrix}$$

$$\Downarrow$$

$$\begin{matrix} 2, & 2, & 2 \\ 2, & 2, & 0 \\ 2, & 0, & 1 \end{matrix}$$



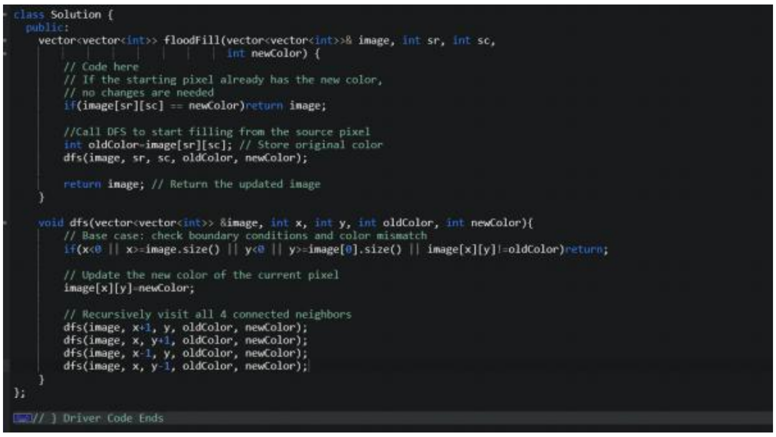
```cpp
// } Driver Code Ends

class Solution {
  public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
                                  int newColor) {
        // Code here
    int n=image.size(), m=image[0].size();
    int og=image[sr][sc];
    vector<vector<int>> dir={{-1,0}, {1,0}, {0,-1}, {0,1}};
    queue<pair<int, int>> q;
    vector<vector<bool>> vis(n, vector<bool>(m,0));
        q.push({sr, sc});
        while(!q.empty()){
            pair<int, int> node=q.front();
            q.pop();
            if(node.first<0 || node.first>=n || node.second<0 || node.second>=m || vis[node.first][node.second] ||
            image[node.first][node.second]!=og)continue;
            vis[node.first][node.second]=1;
            image[node.first][node.second]=newColor;
            for(auto &d:dir){
                q.push({node.first+d[0], node.second+d[1]});
            }
        }


        return image;
    }
};

// } Driver Code Ends
```

← Breadth First search.

T.C. O(nm)

S.C. O(nm) ← queue

Depth First Search :-



```cpp
class Solution {
  public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
                                  int newColor) {
        // Code here
        // If the starting pixel already has the new color,
        // no changes are needed
        if(image[sr][sc] == newColor)return image;

        //Call DFS to start filling from the source pixel
        int oldColor=image[sr][sc]; // Store original color
        dfs(image, sr, sc, oldColor, newColor);

        return image; // Return the updated image
    }

    void dfs(vector<vector<int>> &image, int x, int y, int oldColor, int newColor){
        // Base case: check boundary conditions and color mismatch
        if(x<0 || x>=image.size() || y<0 || y>=image[0].size() || image[x][y]!=oldColor)return;

        // Update the new color of the current pixel
        image[x][y]=newColor;

        // Recursively visit all 4 connected neighbors
        dfs(image, x+1, y, oldColor, newColor);
        dfs(image, x, y+1, oldColor, newColor);
        dfs(image, x-1, y, oldColor, newColor);
        dfs(image, x, y-1, oldColor, newColor);
    }
};

// } Driver Code Ends
```

T.C. O(nm)

S.C. O(nm) ← recursion stack