Flood Fill

An `image` is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate `(sr, sc)` representing the starting pixel (row and column) of the flood fill, and a pixel value `newColor`, "flood fill" the image.

To perform a "flood fill", consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the newColor.

At the end, return the modified image.

Example 1:

```
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.
```

Note:

- The length of `image` and `image[0]` will be in the range `[1, 50]`.
- The given starting pixel will satisfy `0 and 0 .`
- The value of each color in `image[i][j]` and `newColor` will be an integer in `[0, 65535]`.

Solution 1

Time complexity: O(m*n), space complexity: O(1). m is number of rows, n is number of columns.

```
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        if (image[sr][sc] == newColor) return image;
        fill(image, sr, sc, image[sr][sc], newColor);
        return image;
    }

    private void fill(int[][] image, int sr, int sc, int color, int newColor) {
        if (sr < 0 || sr >= image.length || sc < 0 || sc >= image[0].length || image
[sr][sc] != color) return;
        image[sr][sc] = newColor;
        fill(image, sr + 1, sc, color, newColor);
        fill(image, sr - 1, sc, color, newColor);
        fill(image, sr, sc + 1, color, newColor);
        fill(image, sr, sc - 1, color, newColor);
    }
}
```

written by shawngao original link here

Solution 2

The idea is simple. Simply perform a DFS on the source cell. Continue the DFS if:

1. Next cell is within bounds.
2. Next cell is the same color as source cell.

There is a tricky case where the new color is the same as the original color and if the DFS is done on it, there will be an infinite loop. If new color is same as original color, there is nothing to be done and we can simply return the `image`.

- *Yangshun*

```
class Solution(object):
    def floodFill(self, image, sr, sc, newColor):
        rows, cols, orig_color = len(image), len(image[0]), image[sr][sc]
        def traverse(row, col):
            if (not (0 <= row < rows and 0 <= col < cols)) or image[row][col] != or
ig_color:
                return
            image[row][col] = newColor
            [traverse(row + x, col + y) for (x, y) in ((0, 1), (1, 0), (0, -1), (-1
, 0))]
        if orig_color != newColor:
            traverse(sr, sc)
        return image
```

written by yangshun original link here

Solution 3

Java

```
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        if (image[sr][sc] != newColor)
            dfs(image, sr, sc, image[sr][sc], newColor);
        return image;
    }

    private void dfs(int[][] image, int i, int j, int c0, int c1) {
        if (i < 0 || j < 0 || i >= image.length || j >= image[0].length || image[i]
[j] != c0) return;
        image[i][j] = c1;
        dfs(image, i, j - 1, c0, c1);
        dfs(image, i, j + 1, c0, c1);
        dfs(image, i - 1, j, c0, c1);
        dfs(image, i + 1, j, c0, c1);
    }
}
```

C++

```
class Solution {
public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int ne
wColor) {
        if (image[sr][sc] != newColor)
            dfs(image, sr, sc, image[sr][sc], newColor);
        return image;
    }

private:
    void dfs(vector<vector<int>>& image, int i, int j, int c0, int c1) {
        if (i < 0 || j < 0 || i >= image.size() || j >= image[0].size() || image[i]
[j] != c0) return;
        image[i][j] = c1;
        dfs(image, i, j - 1, c0, c1);
        dfs(image, i, j + 1, c0, c1);
        dfs(image, i - 1, j, c0, c1);
        dfs(image, i + 1, j, c0, c1);
    }
};
```

written by alexander original link here