733 Flood Fill

2018年5月4日　　13:35

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 <= sr < image.length and 0 <= sc < image[0].length.
- The value of each color in image[i][j] and newColor will be an integer in [0, 65535].

来自 <https://leetcode.com/problems/flood-fill/description/>

有一幅以二维整数数组表示的图画，每一个整数表示该图画的像素值大小，数值在 0 到 65535 之间。
给你一个坐标 (sr, sc) 表示图像渲染开始的像素值（行，列）和一个新的颜色值 newColor，让你重新上色这幅图像。
为了完成上色工作，从初始坐标开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重复该过程。将所有有记录的像素点的颜色值改为新的颜色值。
最后返回经过上色渲染后的图像。

示例 1:

注意:

- image 和 image[0] 的长度在范围 [1, 50] 内。
- 给出的初始点将满足 0 <= sr < image.length 和 0 <= sc < image[0].length。
- image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535]内。

Solution for Python3:

```python
class Solution1:
    def floodFill(self, image, sr, sc, newColor):
        """
        :type image: List[List[int]]
        :type sr: int
        :type sc: int
        :type newColor: int
        :rtype: List[List[int]]
        """
        R, C, color = len(image), len(image[0]), image[sr][sc]
        if color == newColor:
            return image
        def dfs(r, c):
            if image[r][c] == color:
                image[r][c] = newColor
                if r >= 1:
                    dfs(r-1, c)
                if r+1 < R:
                    dfs(r+1, c)
                if c >= 1:
                    dfs(r, c-1)
                if c+1 < C:
                    dfs(r, c+1)
        dfs(sr, sc)
        return image
```

```python

class Solution2:
    def floodFill(self, image, sr, sc, newColor):
        """
        :type image: List[List[int]]
        :type sr: int
        :type sc: int
        :type newColor: int
        :rtype: List[List[int]]
        """
        R, C, color = len(image), len(image[0]), image[sr][sc]
        def dfs(r, c):
           if (not(0<= r < R and 0 <= c < C)) or image[r][c] != color:
              return
           image[r][c] = newColor
           [dfs(r + i, c + j) for (i, j) in ((0, 1), (0, -1), (1, 0), (-1, 0))]
        if color != newColor:
           dfs(sr, sc)
        return image
```

Solution for C++:

```cpp
class Solution {
public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {
        int color = image[sr][sc];
        if (color != newColor)
            dfs(image, sr, sc, color, newColor);
        return image;
    }
    void dfs(vector<vector<int>>& image, int r, int c, int color, int newColor) {
        if (image[r][c] == color) {
            image[r][c] = newColor;
            if (r >= 1)
                dfs(image, r-1, c, color, newColor);
            if (r+1 < image.size())
                dfs(image, r+1, c, color, newColor);
            if (c >= 1)
                dfs(image, r, c-1, color, newColor);
            if (c+1 < image[0].size())
                dfs(image, r, c+1, color, newColor);
        }
    }
};
```