August 12, 2022 ▪ Graph

Flood Fill Algorithm – Graphs

Problem Statement: An image is represented by a 2-D array of integers, each integer representing the pixel value of the image. 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 colour as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same colour as the starting pixel), and so on. Replace the colour of all of the aforementioned pixels with the newColor.

Search

Search

Recent Posts

Top Array Interview Questions – Structured Path with Video Solutions

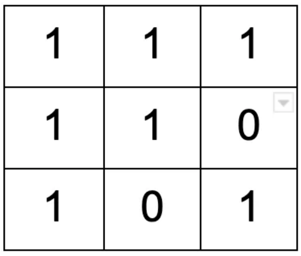
Longest Subarray with sum K | [Postives and Negatives]

Count Subarray sum Equals K

Pre-req: Connected Components, Graph traversal techniques

Example 1:

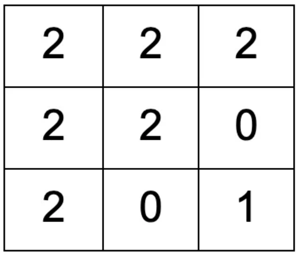
```
Input:
```



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:

Binary Tree Representation in Java

Binary Tree Representation in C++

Accolite Digital Amazon Arcesium arrays Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing infosys inorder Interview Experience Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCS TCS

```
From the centre of the image
(with position (sr, sc) = (1,
1)), all pixels
connected by a path of the same
colour as the
starting pixel are colored with
the new colour.

Note the bottom corner is not
colored 2,
because it is not 4-
directionally connected to
the starting pixel.
```

CODEVITA TCS Ninja

TCS NQT

VMware XOR

Example 2:

```
Input:




sr = 2, sc = 0, newColor = 3

Output:
```

Solution

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Approach:

To perform a “flood fill”, consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same colour as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same colour as the starting pixel), and so on.

We know about both the traversals, Breadth First Search (BFS) and Depth First Search (DFS). We can follow BFS also, where we start at a given point and colour level wise, i.e., we go simultaneously to all its neighbours. We can use any of the traversals to solve this problem, in our case we will be using DFS just to explore multiple approaches.

The algorithm steps are as follows:

- Initial DFS call will start with the starting pixel (sr, sc) and make sure to store the initial colour.
- Now, either we can use the same matrix to replace the colour of all of the aforementioned pixels with the newColor or create a replica of the given matrix. It is advised to use another matrix because we

work on the data and not tamper with it. So we will create a copy of the input matrix.

- Check for the neighbours of the respective pixel that has the same initial colour and has not been visited or coloured. DFS call goes first in the depth on either of the neighbours.
- We go to all 4 directions and check for **unvisited** neighbours with the same initial colour. To travel 4 directions we will use nested loops, you can find the implementation details in the code.
- DFS function call will make sure that it starts the DFS call from that unvisited neighbour, and colours all the pixels that have the same initial colour, and at the same time it will also mark them as visited.

In this way, “flood fill” will be performed. It doesn’t matter how we are colouring the pixels, we just want to colour all of the aforementioned pixels with the newColor. So, we can use any of the traversal techniques.

Consider the following example to understand how DFS traverses the pixels and colours them accordingly.

How to set boundaries for 4 directions?

The 4 neighbours will have following indexes:

Now, either we can apply 4 conditions or follow the following method.

From the above image, it is clear that delta change in row is -1, +0, +1, +0. Similarly, the delta change in column is 0, +1, +0, -1. So we can apply the same logic to find the neighbours of a particular pixel (<row, column>).

Code:

C++ Code

```
#include<bits/stdc++.h>
using namespace std;

class Solution {
private:
    void dfs(int row, int col, vecto
     vector<vector<int>>& image, int
     int iniColor) {
        // color with new color
        ans[row][col] = newColor;
        int n = image.size();
        int m = image[0].size();
        // there are exactly 4 neigh
        for(int i = 0;i<4;i++) {
            int nrow = row + delRow[
            int ncol = col + delCol[
            // check for valid coord
            // then check for same i
            if(nrow>=0 && nrow<n &&
            image[nrow][ncol] == ini
                dfs(nrow, ncol, ans,
            }
        }
    }
public:
    vector<vector<int>> floodFill(ve
    int sr, int sc, int newColor) {
        // get initial color
        int iniColor = image[sr][sc]
        vector<vector<int>> ans = im
        // delta row and delta colum
        int delRow[] = {-1, 0, +1, 0
        int delCol[] = {0, +1, 0, -1
        dfs(sr, sc, ans, image, newC
        return ans;
    }
};
```

```
int main(){

    vector<vector<int>>image{
        {1,1,1},
        {1,1,0},
        {1,0,1}
    };

// sr = 1, sc = 1, newColor = 2
    Solution obj;
    vector<vector<int>> ans = obj.fl
    for(auto i: ans){
        for(auto j: i)
            cout << j << " ";
        cout << "\n";
    }

    return 0;
}
```

Output:

```
2 2 2
2 2 0
2 0 1
```

Time Complexity: O(NxM + NxMx4) ~ O(N x M)

For the worst case, all of the pixels will have the same colour, so DFS function will be called for (N x M) nodes and for every node we are traversing for 4 neighbours, so it will take O(N x M x 4) time.

Space Complexity: O(N x M) + O(N x M)

O(N x M) for copied input array and recursive stack space takes up N x M locations at max.

Java Code

```
import java.util.*;

class Solution
{
    private void dfs(int row, int co
     int[][] ans,
     int[][] image,
     int newColor, int delRow[], int
     int iniColor) {
        // color with new color
        ans[row][col] = newColor;
        int n = image.length;
        int m = image[0].length;
        // there are exactly 4 neigh
        for(int i = 0;i<4;i++) {
            int nrow = row + delRow[
            int ncol = col + delCol[
            // check for valid coord
            // then check for same i
            if(nrow>=0 && nrow<n &&
            image[nrow][ncol] == ini
                dfs(nrow, ncol, ans,
            }
        }
    }
    public int[][] floodFill(int[][]
    {
        // get initial color
        int iniColor = image[sr][sc]
        int[][] ans = image;
        // delta row and delta colum
        int delRow[] = {-1, 0, +1, 0
        int delCol[] = {0, +1, 0, -1
        dfs(sr, sc, ans, image, newC
        return ans;
    }
    public static void main(String[]
    {
        int[][] image =  {
            {1,1,1},
            {1,1,0},
            {1,0,1}
        };
        // sr = 1, sc = 1, newColor
        Solution obj = new Solution(
        int[][] ans = obj.floodFill(
        for(int i = 0; i < ans.lengt
            for(int j = 0; j < ans[i
                System.out.print(ans
            System.out.println();
        }
    }

}
```

Output:

```
2 2 2
2 2 0
2 0 1
```

Time Complexity: O(NxM + NxMx4) ~ O(N x M)

For the worst case, all of the pixels will have the same colour, so DFS function will be called for (N x M) nodes and for every node we are traversing for 4 neighbours, so it will take O(N x M x 4) time.

Space Complexity: O(N x M) + O(N x M)

O(N x M) for copied input array and recursive stack space takes up N x M locations at max.

> Special thanks to **Vanshika Singh Gour** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward

> fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com

« Previous Post
Number of Provinces

Next Post »
Must do Pattern Problems before starting DSA

Load Comments