**Problem Name:** Flood fill

**Topics:**

**Companies:**

**Level:** Easy

**Language:** C++

**Problem Statement**:

Doge is solving a DSA problem but he is having some difficulty with it. Can you help Doge so that he can complete it and play with his friend Cheem?

Problem is:

**Input Format:**

The first line of input is an integer value n (total no of vertex in the graph).

The Second line of input contains integer value m (total no of edges in the graph)

The next m lines contain two space-separated integers forming a graph.

Last line of input contains the integer value source and destination

**Output Format:**

**Constraints:**

**Examples:**

**Approach one Solution:**

**Explanation:** This simple algorithm can be easily extended:

* togo list, safe and clean
* seen matrix to avoid loops
* dirs to loop over all possible directions
* process togo till it's empty and add elements to that queue as you find them

If that desription was too fast, then here is a longer version:

* keep places to go in a queue, togo, e.g. vector<T> togo. I will explain what is T later.
* keep track of visited places in a matrix of the same size as original: vector<vector<int>> seen. You can call it visited, but usually it leads to more typos. This helps to never go to the same place twice
* keep the queue safe and clean, then the queue will be shorter and will never have duplicates; also you will not waste time on adding elems to the queue if they are not valid saving time and memory
* walk through the queue (list) and add elements to be checked  
  It's like hunting for mushrooms: you take one, see several others at the same time and add them to your "togo" list. Keep gooing till you get all of them.

**Code:**

**Time Complexity**:

**Space Complexity:**

**Approach second Solution:**

Explanation: DFS

Function wise Explanation:

**1. floodFil():**

1. get the pixel color at the given coordinates sr, sc
2. if the old color and new color is not same call dfs function.

**2. dfs()**

1. a. check the boundray conditions for running dfs ( i,j <0 --> this means, dfs should run, image array lowerbound, i >= image.size() and j >= image[0].size() for upper bound.)  
   b. next pixel color should match with actual color(**oldColor**) of the current pixel.
2. set the current pixel color to new color
3. run the dfs for  
   a. upperpixel  
   b. lower pixel  
   c. left pixel  
   d. right pixel to current pixel.

**Code:**

**Time Complexity**: No.of Pixels i.e., O(N)

**Space Complexity:** No of dfs calls(Nothing but no.of pixels in worsecase) O(N).