Connecting Dots

Problem Level: Easy

Problem Description:

Gary has a board of size NxM. Each cell in the board is a coloured dot. There exist only 26 colours denoted by uppercase Latin characters (i.e. A,B,...,Z). Now Gary is getting bored and wants to play a game. The key of this game is to find a cycle that contains dots of the same colour. Formally, we call a sequence of dots d1, d2, ..., dk a cycle if and only if it meets the following condition:
1. These k dots are different: if i ≠ j then di is different from dj.
2. k is at least 4.
3. All dots belong to the same colour.
4. For all 1 ≤ i ≤ k - 1: di and di + 1 are adjacent. Also, dk and d1 should also be adjacent. Cells x and y are called adjacent if they share an edge.

Since Gary is colour blind, he wants your help. Your task is to determine if there exists a cycle on the board.

Sample Input 1:

```
3 4
AAAA
ABCA
AAAA
```

Sample Output 1:

```
true
```

Approach to be followed:

Our goal is to find whether a cycle where 4 or more dots connected satisfy the above mentioned conditions or not. For this, we may easily use the concept of recursion. We shall maintain a visited 2D array, which will keep track of the visited dots. Now, for each dot, we will recurse in all four directions, and keep on recursing if we find any dot with the same color in any of the four directions. If while performing recursion, we reach back to the original dot, that is, if the visited

2D array for some point is found to be true, this means that through continuous recursion, we have reached to the point we began from, while satisfying the conditions of the same color.

Hence, a cycle with the mentioned conditions exists, and we return true.

Steps:

1. For each color on the board, recurse in all four directions, if and only if the current color is in bounds of the board, and is the same as the previous color.
2. Maintain a boolean variable **findCycle**, to keep check if the cycle is found or not.
3. Mark the visited 2D array for the current position as true.
4. Using a loop, call recursion in all four directions.
5. While performing DFS of the board, if visited becomes true, mark findCycle as true and return.

Pseudo Code:

```
dx = {1, -1, 0, 0}
dy = {0, 0, 1, -1}
declare visited
findCycle = false

function dfs(board, x, y, fromX, fromY, needColor, n, m)
    if x < 0 or x >= n or y < 0 or y >= m
        return
    if board[x][y] not equal to needColor
        return
    if visited[x][y] equals true
        findCycle = true
        return

    visited[x][y] = true
    loop for f = 0 till f = 4
        nextX = x + dx[f]
```

```
          nextY = y + dy[f]

          if nextX equals fromX and nextY equals fromY

              continue

          dfs(board, nextX, nextY, x, y, needColor, n, m);


function solve(board, n, m)

    fill visited with false

    loop from i = 0 till i = n

        loop from j = 0 till j = m

            if visited[i][j] is false

                dfs(board,i, j, -1, -1, board[i][j], n, m)


    return findCycle
```

Time Complexity: O(N * M), where **N** and **M** are the dimensions of the board. In the worst case, we will perform recursion for each and every dot, hence overall time complexity is **O(N * M)**