Daily Coding Problem #151

Problem

Given a 2-D matrix representing an image, a location of a pixel in the screen and a color C, replace the color of the given pixel and all adjacent same colored pixels with C.

For example, given the following matrix, and location pixel of (2, 2), and 'G' for green:

```
B B W
W W W
W W W
B B B
```

Becomes

```
B B G
G G G
G G G
B B B
```

Solution

A simplistic strategy to floodfill is to use depth-first-search. The algorithm works as follows:

First, Fill the current `coord` to `color`. Mark as visited. For each neighboring `new_coord`, if it hasn't been visited, is inside the matrix, and is the same color as current `coord` color, recursively floodfill that coordinate.

```
def floodfill(matrix, coord, color, visited=None):
    if visited is None:
        visited = set()

    visited.add(coord)

    r, c = coord
    prior_color = matrix[r][c]
    matrix[r][c] = color

    coords = [(r + 1, c), (r, c + 1), (r - 1, c), (r, c - 1)]

    for new_coord in coords:
        new_r, new_c = new_coord
        if (new_coord not in visited
            and in_matrix(matrix, new_coord)
            and matrix[new_r][new_c] == prior_color):

            visited.add(new_coord)
            floodfill(matrix, new_coord, color, visited)


def in_matrix(matrix, coord):
    rows = len(matrix)
    cols = len(matrix[0])
```

```
    r, c = coord

    return 0 <= r < rows and 0 <= c < cols
```

This will take O(V + E) time and O(V) space. In other words, it will take time proportional to the size of the matrix.

Another way to do this problem would be to use breadth first search, using a queue. This would also yield the same complexities.
