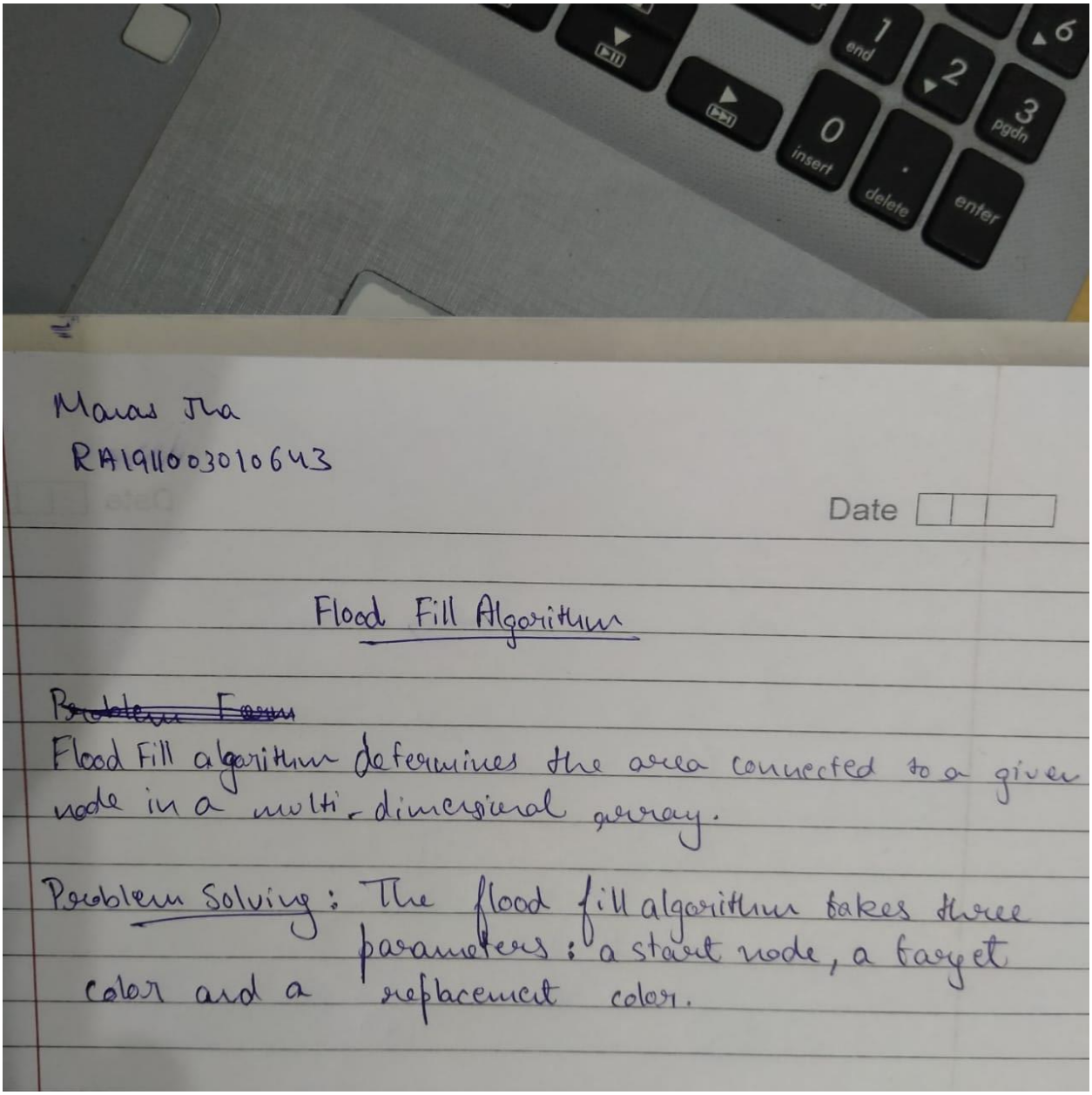
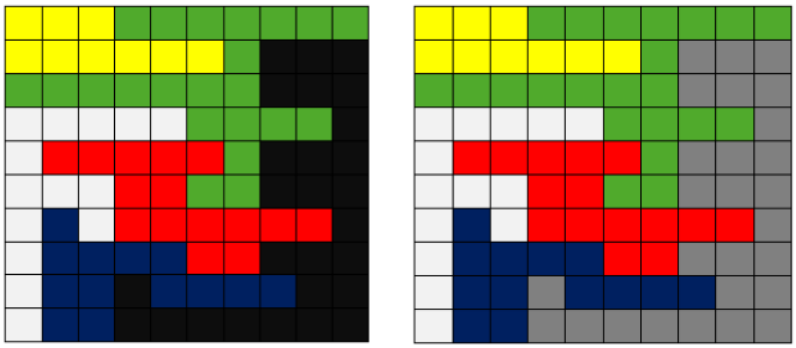
Manas Jha

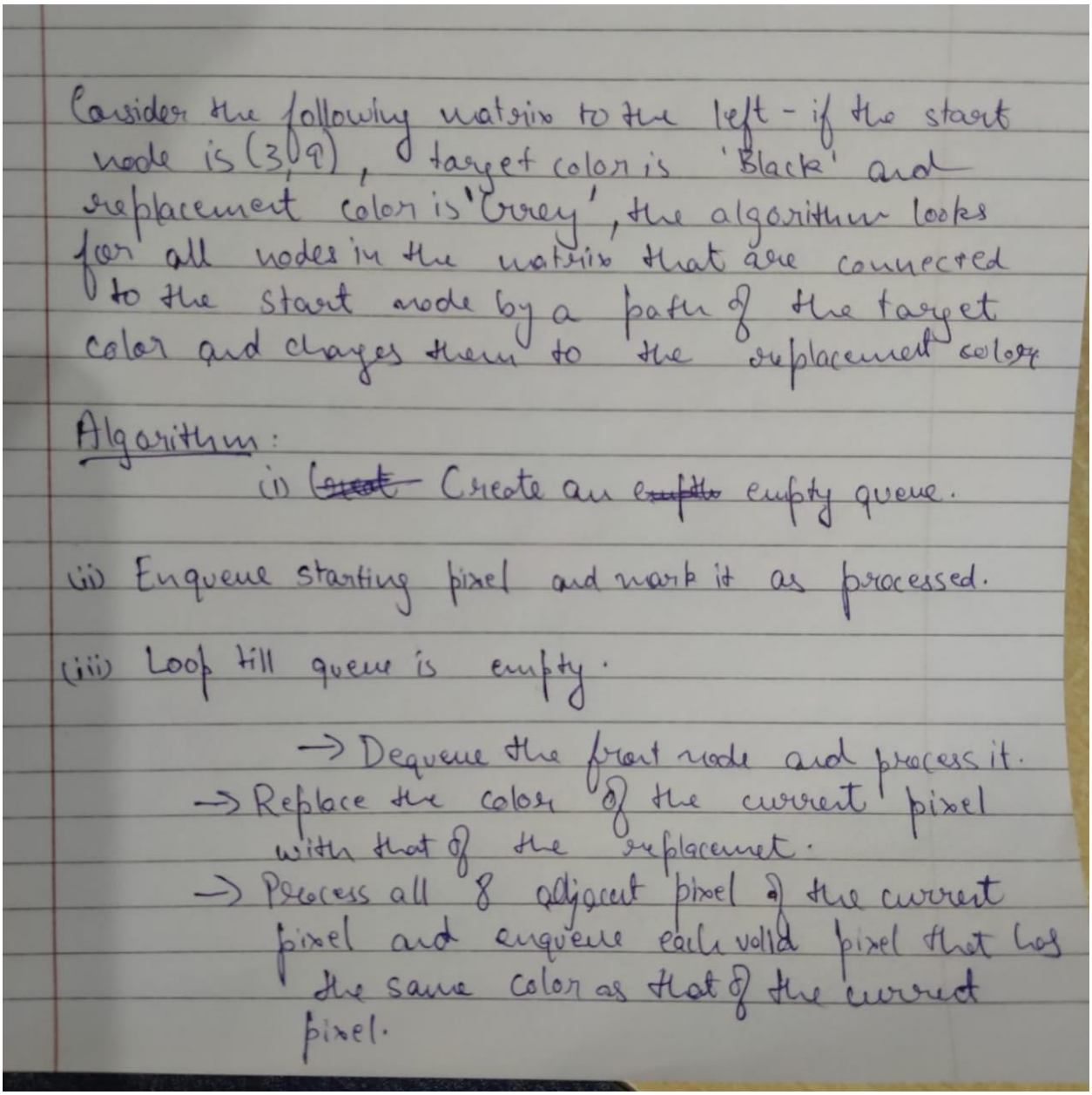
RA1911003010643

Flood Fill Algorithm





Note that each cell of the matrix represents one pixel.

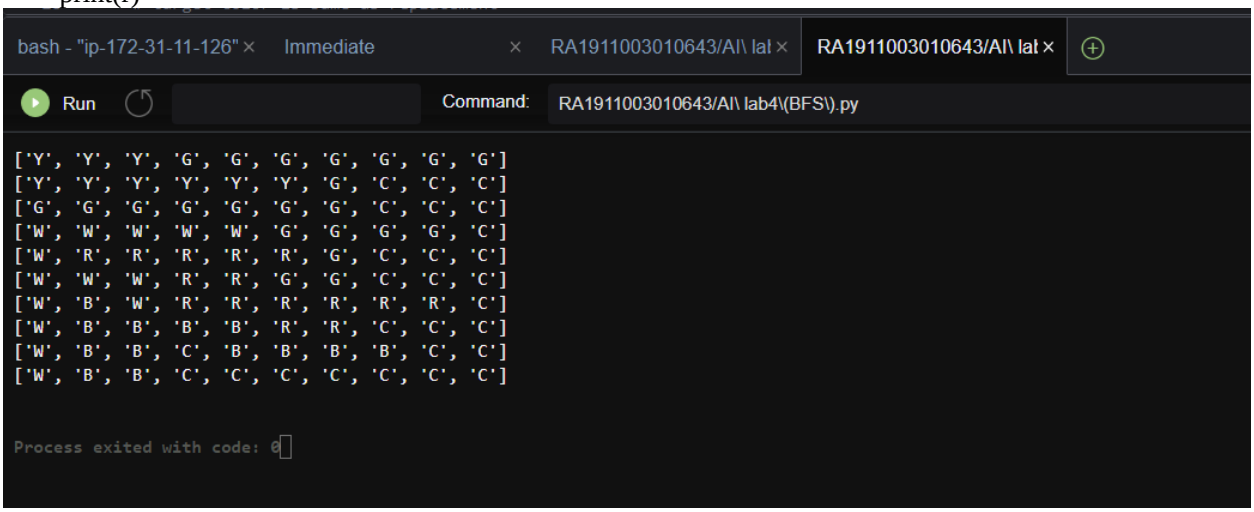


BFS

```
from collections import deque


# Below lists detail all eight possible movements
row = [-1, -1, -1, 0, 0, 1, 1, 1]
col = [-1, 0, 1, -1, 1, -1, 0, 1]


# check if it is possible to go to pixel (x, y) from the
# current pixel. The function returns false if the pixel
```

```
# has a different color, or it's not a valid pixel
def isSafe(mat, x, y, target):
    return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and mat[x][y] == target


# Flood fill using BFS
def floodfill(mat, x, y, replacement):

    # base case
    if not mat or not len(mat):
        return

    # create a queue and enqueue starting pixel
    q = deque()
    q.append((x, y))

    # get the target color
    target = mat[x][y]

    # target color is same as replacement
    if target == replacement:
        return

    # break when the queue becomes empty
    while q:

        # dequeue front node and process it
        x, y = q.popleft()

        # replace the current pixel color with that of replacement
        mat[x][y] = replacement

        # process all eight adjacent pixels of the current pixel and
        # enqueue each valid pixel
        for k in range(len(row)):
            # if the adjacent pixel at position (x + row[k], y + col[k]) is
            # is valid and has the same color as the current pixel
            if isSafe(mat, x + row[k], y + col[k], target):
                # enqueue adjacent pixel
                q.append((x + row[k], y + col[k]))


if __name__ == '__main__':

    # matrix showing portion of the screen having different colors
    mat = [
            ['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G'],
            ['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'X', 'X', 'X'],
            ['G', 'G', 'G', 'G', 'G', 'G', 'G', 'X', 'X', 'X'],
            ['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'X'],
            ['W', 'R', 'R', 'R', 'R', 'R', 'G', 'X', 'X', 'X'],
            ['W', 'W', 'W', 'R', 'R', 'G', 'G', 'X', 'X', 'X'],
            ['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'X'],
```

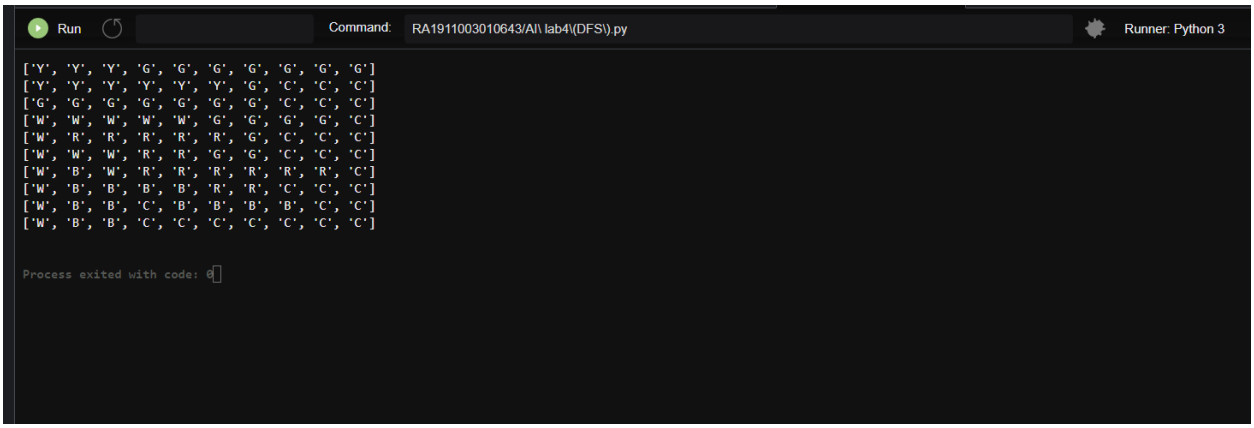
```
        ['W', 'B', 'B', 'B', 'B', 'R', 'R', 'X', 'X', 'X'],
        ['W', 'B', 'B', 'X', 'B', 'B', 'B', 'B', 'X', 'X'],
        ['W', 'B', 'B', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
    ]

    # start node
    x = 3
    y = 9

    # having target color `X`
    # replacement color
    replacement = 'C'

    # replace the target color with a replacement color
    floodfill(mat, x, y, replacement)

    # print the colors after replacement
    for r in mat:
        print(r)
```



DFS

```
# Below lists detail all eight possible movements
row = [-1, -1, -1, 0, 0, 1, 1, 1]
col = [-1, 0, 1, -1, 1, -1, 0, 1]


# check if it is possible to go to pixel (x, y) from the
# current pixel. The function returns false if the pixel
# has a different color, or it's not a valid pixel
def isSafe(mat, x, y, target):
    return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and mat[x][y] == target


# Flood fill using DFS
def floodfill(mat, x, y, replacement):

    # base case
```

```
    if not mat or not len(mat):
        return

    # get the target color
    target = mat[x][y]

    # target color is same as replacement
    if target == replacement:
        return

    # replace the current pixel color with that of replacement
    mat[x][y] = replacement

    # process all eight adjacent pixels of the current pixel and
    # recur for each valid pixel
    for k in range(len(row)):

        # if the adjacent pixel at position (x + row[k], y + col[k]) is
        # a valid pixel and has the same color as that of the current pixel
        if isSafe(mat, x + row[k], y + col[k], target):
            floodfill(mat, x + row[k], y + col[k], replacement)


if __name__ == '__main__':

    # matrix showing portion of the screen having different colors
    mat = [
            ['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G'],
            ['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'X', 'X', 'X'],
            ['G', 'G', 'G', 'G', 'G', 'G', 'G', 'X', 'X', 'X'],
            ['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'X'],
            ['W', 'R', 'R', 'R', 'R', 'R', 'G', 'X', 'X', 'X'],
            ['W', 'W', 'W', 'R', 'R', 'G', 'G', 'X', 'X', 'X'],
            ['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'X'],
            ['W', 'B', 'B', 'B', 'B', 'R', 'R', 'X', 'X', 'X'],
            ['W', 'B', 'B', 'X', 'B', 'B', 'B', 'B', 'X', 'X'],
            ['W', 'B', 'B', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
        ]

    # start node
    x, y = (3, 9)   # having a target color `X`

    # replacement color
    replacement = 'C'

    # replace the target color with a replacement color using DFS
    floodfill(mat, x, y, replacement)

    # print the colors after replacement
    for r in mat:
        print(r)
```



The time complexity of the proposed solution is $O(M \times N)$ and requires $O(M \times N)$ extra space, where `M` and `N` are dimensions of the matrix.

-

-