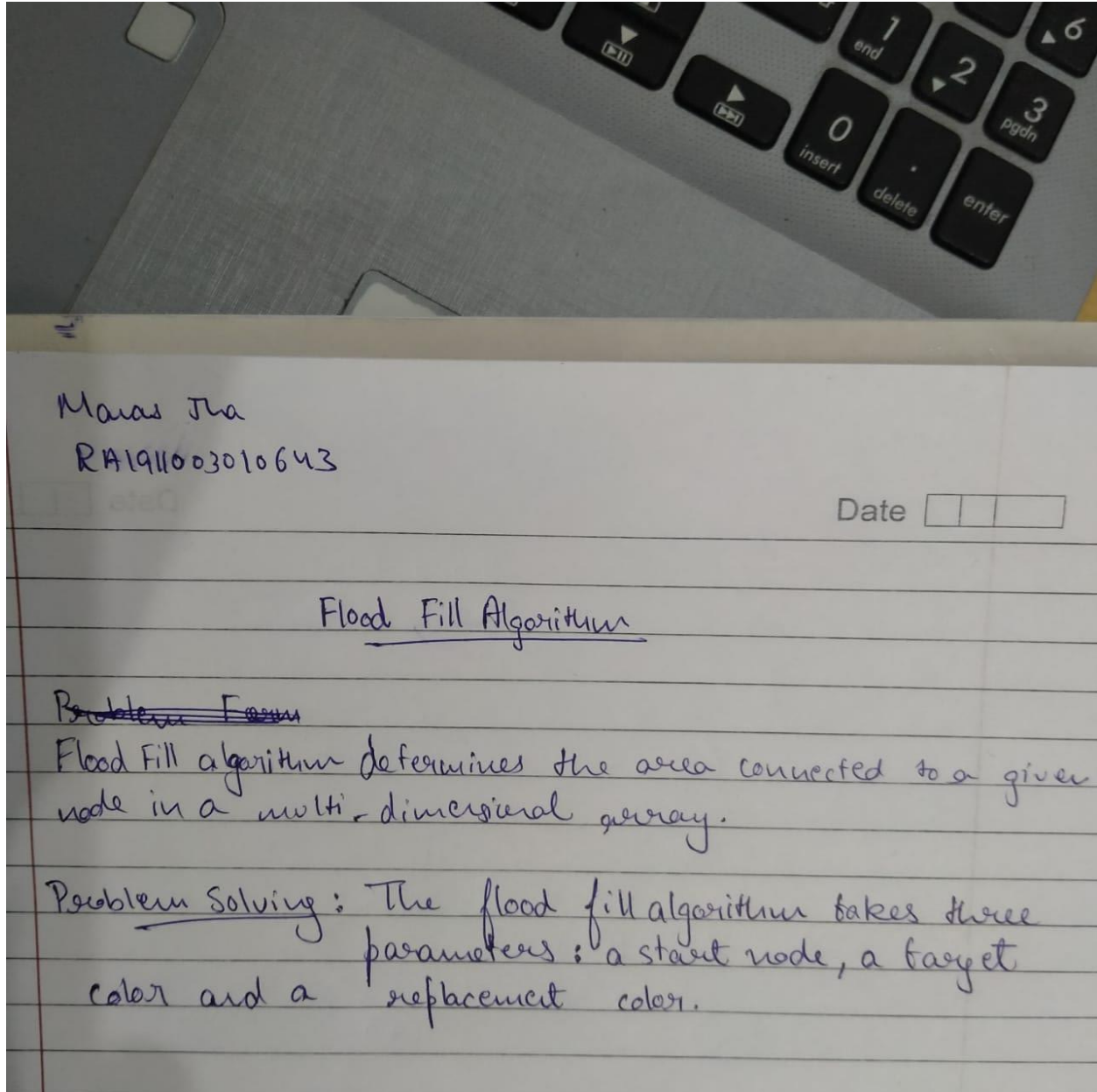
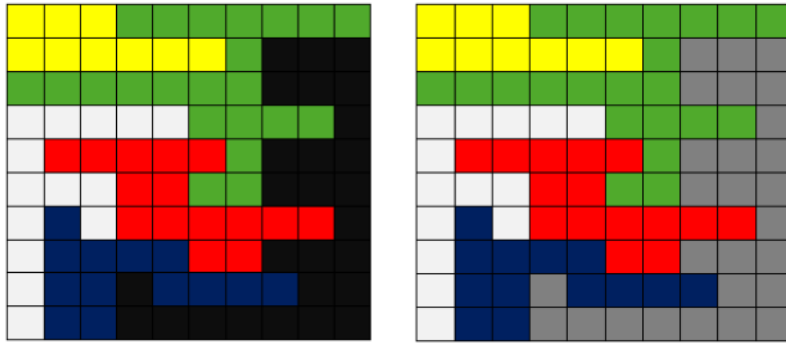


Manas Jha

RA1911003010643

Flood Fill Algorithm





Note that each cell of the matrix represents one pixel.

Consider the following matrix to the left - if the start node is (3,9), target color is 'Black' and replacement color is 'Grey', the algorithm looks for all nodes in the matrix that are connected to the start node by a path of the target color and changes them to the replacement color.

Algorithm:

- (i) ~~Create~~ Create an ~~empty~~ empty queue.
- (ii) Enqueue starting pixel and mark it as processed.
- (iii) Loop till queue is empty.
 - Dequeue the front node and process it.
 - Replace the color of the current pixel with that of the replacement.
 - Process all 8 adjacent pixel of the current pixel and enqueue each valid pixel that has the same color as that of the current 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)

```

```

bash - "ip-172-31-11-126" x Immediate x RA1911003010643/Al\ lat x RA1911003010643/Al\ lat x
Run Command: RA1911003010643/Al\ lab4\BFS\)'
['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'C', 'C', 'C']
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'C']
['W', 'R', 'R', 'R', 'R', 'R', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'R', 'R', 'G', 'G', 'C', 'C', 'C']
['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'C']
['W', 'B', 'B', 'B', 'B', 'R', 'R', 'C', 'C', 'C']
['W', 'B', 'B', 'C', 'B', 'B', 'B', 'B', 'C', 'C']
['W', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'C']
Process exited with code: 0

```

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'],
        ['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'X', 'X'],
        ['G', 'G', 'G', 'G', 'G', 'G', 'X', 'X', 'X'],
        ['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'X'],
        ['W', 'R', 'R', 'R', 'R', 'R', 'G', 'X', 'X'],
        ['W', 'W', 'W', 'R', 'R', 'G', 'G', 'X', 'X'],
        ['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'X'],
        ['W', 'B', 'B', 'B', 'B', 'R', 'R', 'X', 'X'],
        ['W', 'B', 'B', 'X', 'B', 'B', 'B', 'X', 'X'],
        ['W', 'B', 'B', '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)

```

```
Run Command: RA1911003010643/AI\ lab4\DFS\).py Runner: Python 3

['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G']
['Y', 'Y', 'Y', 'Y', 'Y', 'G', 'C', 'C', 'C']
['G', 'G', 'G', 'G', 'G', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'C']
['W', 'R', 'R', 'R', 'R', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'R', 'R', 'G', 'C', 'C', 'C']
['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'C']
['W', 'B', 'B', 'B', 'B', 'R', 'C', 'C', 'C']
['W', 'B', 'B', 'C', 'B', 'B', 'B', 'C', 'C']
['W', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'C']

Process exited with code: 0
```

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.

-

-