**Approach 1: BruteForce Approach/ extra space**

1) for each cell, apply the rules by finding state of all 8 neighbours, and store the new value in a new array,

2) copy values from new array to old array

Time Complexity => O(m\*n) bfs = O(k=4)

Space COmplexity => O(m\*n)

**Code:**

r = len(board)

c = len(board[0])

# create new empty array

new\_board = [[0 for i in range(c)] for j in range(r)]

def bfs(i,j):

nei = [(i-1,j), (i,j-1), (i,j+1), (i+1,j), (i-1,j-1), (i+1,j+1),(i-1,j+1),(i+1,j-1)]

live = 0

dead = 0

for n in nei:

x,y = n

if x>=0 and y>=0 and x<r and y<c:

if board[x][y] == 0:

dead += 1

else:

live +=1

# set

if board[i][j] == 1:

if live < 2 or live > 3:

new\_board[i][j] = 0

else:

new\_board[i][j] = 1

else:

if live ==3:

new\_board[i][j] = 1

for i in range(r):

for j in range(c):

bfs(i,j)

for i in range(r):

for j in range(c):

board[i][j] = new\_board[i][j]

**Approach 2: Space efficient/ Use of flags**

| **Old** | **New** | **Interpreted** | **Final** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 3 | 1 |

0 1 2 3 : Old meaning 0 means 0 remained 0

2 means 0 changed to 1

1 means 1 changed to 0

3 means 1 remained 1

**Note that** : the change in 1 and 2 because if we use the other logic that is 0 1 => 1

Then changed/ interpreted value that is 1 means it denotes that previously this was a dead cell but if 1 denotes a dead cell, leads to confusions. Therefore restore meaning of 0 as 0 by marking 0 to 1 transition as 2

**Code**

r = len(board)

c = len(board[0])

# create new empty array

new\_board = [[0 for i in range(c)] for j in range(r)]

def bfs(i,j):

nei = [(i-1,j), (i,j-1), (i,j+1), (i+1,j), (i-1,j-1), (i+1,j+1),(i-1,j+1),(i+1,j-1)]

live = 0

dead = 0

for n in nei:

x,y = n

if x>=0 and y>=0 and x<r and y<c:

if board[x][y] == 0 or board[x][y] == 2:

dead += 1

else:

live +=1

# set

if board[i][j] == 1:

if live < 2 or live > 3:

board[i][j] = 1

#print(f"i={i} j={j} newboard1 = {new\_board[i][j]}")

else:

board[i][j] = 3

else:

if live ==3:

board[i][j] = 2

#print(f"newboard = {new\_board[i][j]}")

#print(f"live={live} dead={dead} i={i} j={j} {new\_board[i][j]}")

for i in range(r):

for j in range(c):

bfs(i,j)

# update to final values

for i in range(r):

for j in range(c):

if board[i][j] == 1:

board[i][j] = 0

elif board[i][j] == 2 or board[i][j] == 3:

board[i][j] = 1