Image Smoother

Given a 2D integer matrix M representing the gray scale of an image, you need to design a smoother to make the gray scale of each cell becomes the average gray scale (rounding down) of all the 8 surrounding cells and itself. If a cell has less than 8 surrounding cells, then use as many as you can.

Example 1:

```
Input:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
Output:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
Explanation:
For the point (0,0), (0,2), (2,0), (2,2): floor(3/4) = floor(0.75) = 0
For the point (0,1), (1,0), (1,2), (2,1): floor(5/6) = floor(0.83333333) = 0
For the point (1,1): floor(8/9) = floor(0.88888889) = 0
```

Note:

1. The value in the given matrix is in the range of [0, 255].
2. The length and width of the given matrix are in the range of [1, 150].

Solution 1

Here we have a check function to check the boundary and a inner double loop to traverse the 9 potential candidates:

```
public class ImageSmoother {

    public int[][] imageSmoother(int[][] M) {
        if (M == null) return null;
        int rows = M.length;
        if (rows == 0) return new int[0][];
        int cols = M[0].length;

        int result[][] = new int[rows][cols];

        for (int row = 0; row < rows; row++) {
            for (int col = 0; col < cols; col++) {
                int count = 0;
                int sum = 0;
                for (int incR : new int[]{-1, 0, 1}) {
                    for (int incC : new int[]{-1, 0, 1}) {
                        if (isValid(row + incR, col + incC, rows, cols)) {
                            count++;
                            sum += M[row + incR][col + incC];
                        }
                    }
                }
                result[row][col] = sum / count;
            }
        }

        return result;

    }

    private boolean isValid(int x, int y, int rows, int cols) {
        return x >= 0 && x < rows && y >= 0 && y < cols;
    }
}
```

written by seanzhou1023 original link here

Solution 2

Check the numbers around on valid indices and increment the sum and quotient(count) for average whenever you find a valid occurrence.

```
public int[][] imageSmoother(int[][] M) {
        int[][] res = new int[M.length][M[0].length];
        int count = 0;
        int sum = 0;
        for(int i = 0 ; i < M.length ; i++){

            for(int j = 0 ; j < M[0].length ; j++){
                sum =M[i][j];
                count=1;

                if(i-1>=0){
                    sum+=M[i-1][j];
                    count++;
                    if(j-1>=0){
                        sum+=M[i-1][j-1];
                        count++;
                    }
                    if(j+1<M[0].length){
                        sum+=M[i-1][j+1];
                        count++;
                    }
                }

                if(j+1<M[0].length){
                    sum+=M[i][j+1];
                    count++;
                }

                if(j-1>=0){
                    sum+=M[i][j-1];
                    count++;
                    if(i+1<M.length){
                        sum+=M[i+1][j-1];
                        count++;
                    }
                }

                if(i+1<M.length){
                    sum+=M[i+1][j];
                    count++;

                    if(j+1<M[0].length){
                        sum+=M[i+1][j+1];
                        count++;
                    }
                }
                res[i][j] = (int)Math.floor(sum/count);
            }
        }

        return res;
    }
```

written by waltwhitman original link here

Solution 3

```
from copy import deepcopy as copy

class Solution(object):
    def imageSmoother(self, M):
        """
        :type M: List[List[int]]
        :rtype: List[List[int]]
        """
        x_len = len(M)
        y_len = len(M[0]) if x_len else 0
        res = copy(M)
        for x in range(x_len):
            for y in range(y_len):
                neighbors = [
                    M[_x][_y]
                    for _x in (x-1, x, x+1)
                    for _y in (y-1, y, y+1)
                    if 0 <= _x < x_len and 0 <= _y < y_len
                ]
                res[x][y] = sum(neighbors) // len(neighbors)
        return res
```

written by niksite original link here