Problem 661: Image Smoother

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

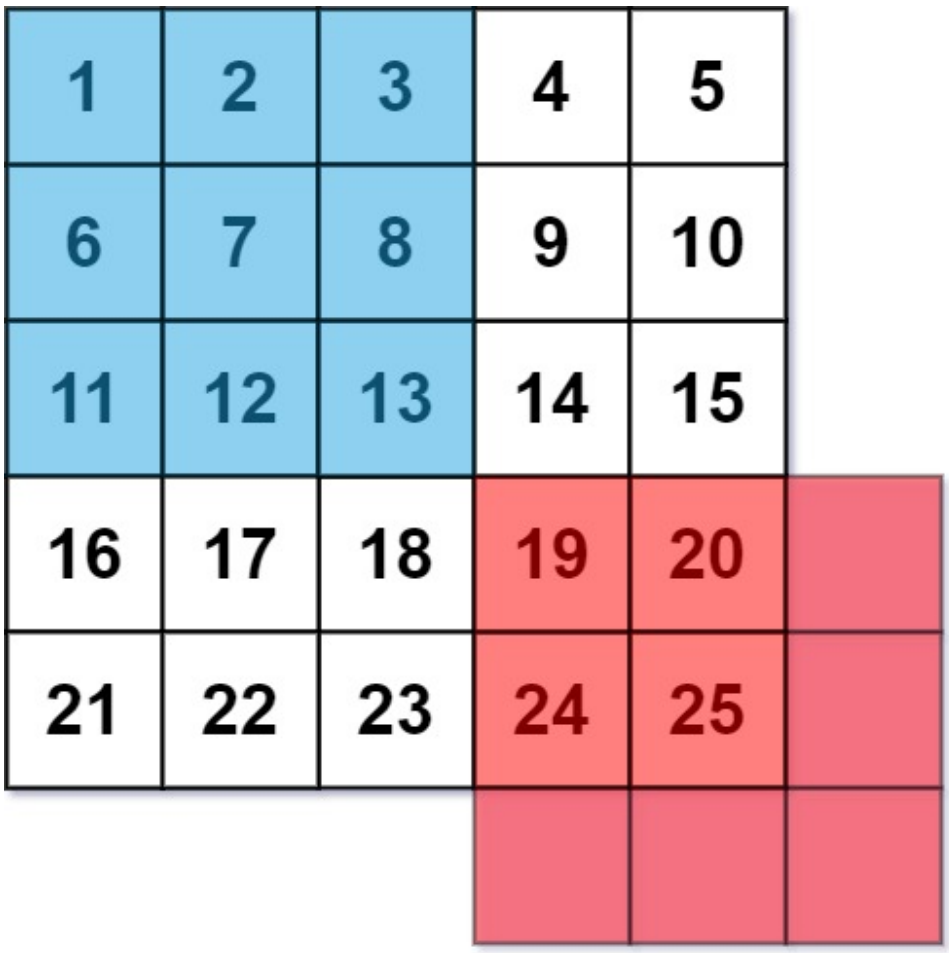
An

image smoother

is a filter of the size

3 x 3

that can be applied to each cell of an image by rounding down the average of the cell and the eight surrounding cells (i.e., the average of the nine cells in the blue smoother). If one or more of the surrounding cells of a cell is not present, we do not consider it in the average (i.e., the average of the four cells in the red smoother).



Given an

m x n

integer matrix

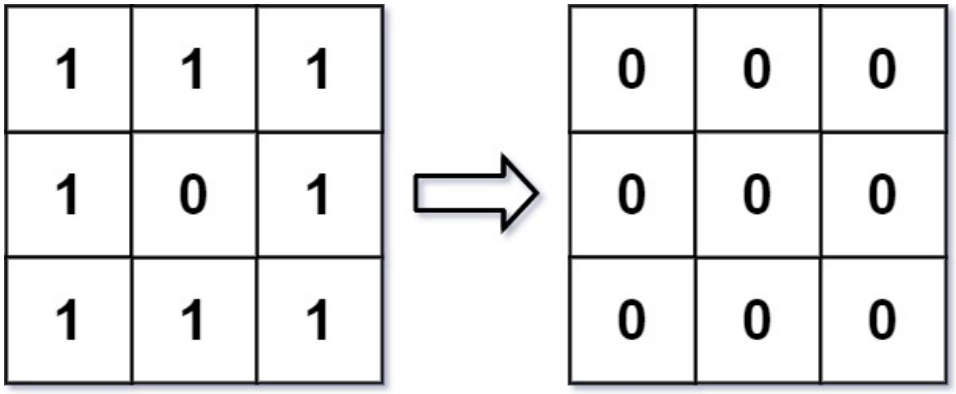
img

representing the grayscale of an image, return

the image after applying the smoother on each cell of it

.

Example 1:



Input:

img = [[1,1,1],[1,0,1],[1,1,1]]

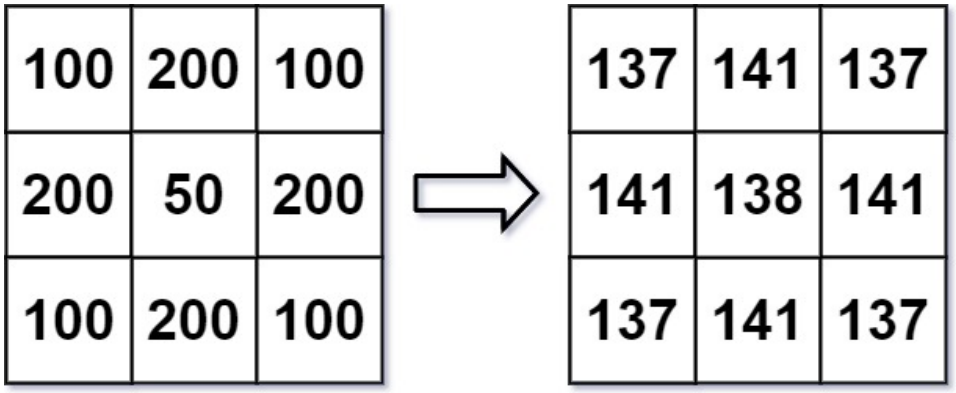
Output:

[[0,0,0],[0,0,0],[0,0,0]]

Explanation:

For the points (0,0), (0,2), (2,0), (2,2): floor(3/4) = floor(0.75) = 0 For the points (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

Example 2:



Input:

img = [[100,200,100],[200,50,200],[100,200,100]]

Output:

[[137,141,137],[141,138,141],[137,141,137]]

Explanation:

For the points (0,0), (0,2), (2,0), (2,2): floor((100+200+200+50)/4) = floor(137.5) = 137 For the points (0,1), (1,0), (1,2), (2,1): floor((200+200+50+200+100+100)/6) = floor(141.666667) = 141 For the point (1,1): floor((50+200+200+200+200+100+100+100+100)/9) = floor(138.888889) = 138

Constraints:

m == img.length

n == img[i].length

1 <= m, n <= 200

0 <= img[i][j] <= 255

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> imageSmoother(vector<vector<int>>& img) {

}
};
```

Java:

```
class Solution {
public int[][] imageSmoother(int[][] img) {

}
}
```

Python3:

```
class Solution:
def imageSmoother(self, img: List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def imageSmoother(self, img):
"""
:type img: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} img
* @return {number[][]}
*/
var imageSmoother = function(img) {

};
```

TypeScript:

```
function imageSmoother(img: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ImageSmoother(int[][] img) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** imageSmoother(int** img, int imgSize, int* imgColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func imageSmoother(img [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun imageSmoother(img: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func imageSmoother(_ img: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn image_smoother(img: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} img
# @return {Integer[][]}
```

```
def image_smoother(img)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $img
* @return Integer[][]
*/
function imageSmoother($img) {

}
}
```

Dart:

```
class Solution {
List<List<int>> imageSmoother(List<List<int>> img) {

}
}
```

Scala:

```
object Solution {
def imageSmoother(img: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec image_smoother(img :: [[integer]]) :: [[integer]]
def image_smoother(img) do

end
end
```

Erlang:

```
-spec image_smoother(Img :: [[integer()]]) -> [[integer()]].
image_smoother(Img) ->
.
```

Racket:

```
(define/contract (image-smoother img)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Image Smoother
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> imageSmoother(vector<vector<int>>& img) {

}
};
```

Java Solution:

```
/**
* Problem: Image Smoother
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] imageSmoother(int[][] img) {

}
}
```

Python3 Solution:

```
"""
Problem: Image Smoother
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def imageSmoother(self, img: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def imageSmoother(self, img):
"""
:type img: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Image Smoother
* Difficulty: Easy
```

```
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} img
* @return {number[][]}
*/
var imageSmoother = function(img) {

};
```

TypeScript Solution:

```
/**
* Problem: Image Smoother
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function imageSmoother(img: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Image Smoother
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public int[][] ImageSmoother(int[][] img) {

}
}
```

C Solution:

```
/*
* Problem: Image Smoother
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** imageSmoother(int** img, int imgSize, int* imgColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Image Smoother
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func imageSmoother(img [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun imageSmoother(img: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func imageSmoother(_ img: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Image Smoother
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn image_smoother(img: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} img
# @return {Integer[][]}
def image_smoother(img)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $img
* @return Integer[][]
*/
function imageSmoother($img) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> imageSmoother(List<List<int>> img) {

}
}
```

Scala Solution:

```
object Solution {
def imageSmoother(img: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec image_smoother(img :: [[integer]]) :: [[integer]]
def image_smoother(img) do

end
end
```

Erlang Solution:

```
-spec image_smoother(Img :: [[integer()]]) -> [[integer()]].
image_smoother(Img) ->
.
```

Racket Solution:

```
(define/contract (image-smoother img)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```