Problem 3030: Find the Grid of Region Average

Problem Information

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

Problem Description

You are given

m x n

grid

image

which represents a grayscale image, where

image[i][j]

represents a pixel with intensity in the range

[0..255]

. You are also given a

non-negative

integer

threshold

.

Two pixels are

adjacent

if they share an edge.

A

region

is a

3 x 3

subgrid where the

absolute difference

in intensity between any two

adjacent

pixels is

less than or equal to

threshold

.

All pixels in a region belong to that region, note that a pixel can belong to

multiple

regions.

You need to calculate a

m x n

grid

result

, where

result[i][j]

is the

average

intensity of the regions to which

image[i][j]

belongs,

rounded down

to the nearest integer. If

image[i][j]

belongs to multiple regions,

result[i][j]

is the

average

of the

rounded-down average

intensities of these regions,

rounded down

to the nearest integer. If

image[i][j]

does

not

belong to any region,

result[i][j]

is

equal to

image[i][j]

.

Return the grid

result

.

Example 1:

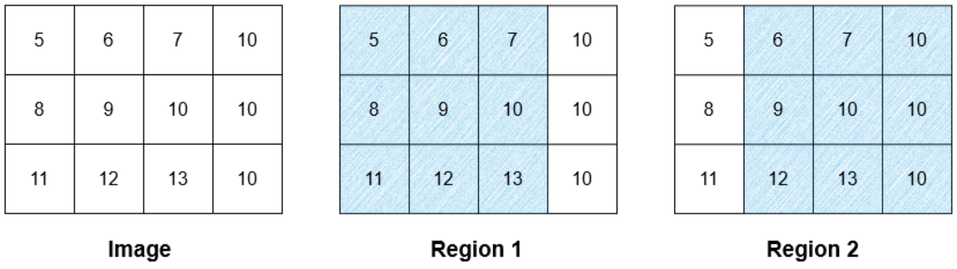
Input:

image = [[5,6,7,10],[8,9,10,10],[11,12,13,10]], threshold = 3

Output:

[[9,9,9,9],[9,9,9,9],[9,9,9,9]]

Explanation:



There are two regions as illustrated above. The average intensity of the first region is 9, while the average intensity of the second region is 9.67 which is rounded down to 9. The average intensity of both of the regions is (9 + 9) / 2 = 9. As all the pixels belong to either region 1, region 2, or both of them, the intensity of every pixel in the result is 9.

Please note that the rounded-down values are used when calculating the average of multiple regions, hence the calculation is done using 9 as the average intensity of region 2, not 9.67.

Example 2:

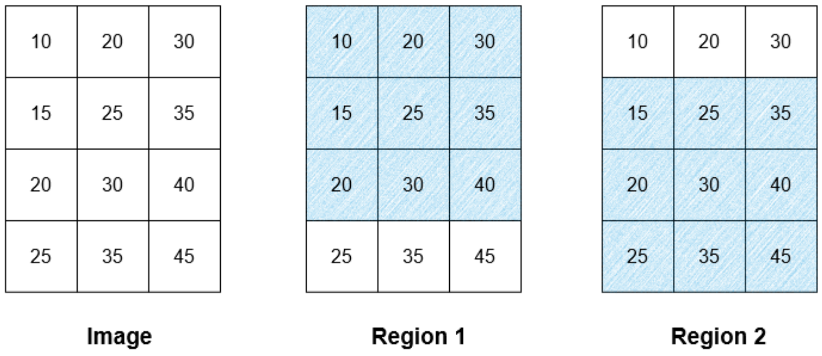
Input:

image = [[10,20,30],[15,25,35],[20,30,40],[25,35,45]], threshold = 12

Output:

[[25,25,25],[27,27,27],[27,27,27],[30,30,30]]

Explanation:



There are two regions as illustrated above. The average intensity of the first region is 25, while the average intensity of the second region is 30. The average intensity of both of the regions is (25 + 30) / 2 = 27.5 which is rounded down to 27.

All the pixels in row 0 of the image belong to region 1, hence all the pixels in row 0 in the result are 25. Similarly, all the pixels in row 3 in the result are 30. The pixels in rows 1 and 2 of the image belong to region 1 and region 2, hence their assigned value is 27 in the result.

Example 3:

Input:

image = [[5,6,7],[8,9,10],[11,12,13]], threshold = 1

Output:

[[5,6,7],[8,9,10],[11,12,13]]

Explanation:

There is only one

3 x 3

subgrid, while it does not have the condition on difference of adjacent pixels, for example, the difference between

image[0][0]

and

image[1][0]

is

|5 - 8| = 3 > threshold = 1

. None of them belong to any valid regions, so the result

should be the same as

image

.

Constraints:

3 <= n, m <= 500

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

0 <= threshold <= 255

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> resultGrid(vector<vector<int>>& image, int threshold) {

}
};
```

Java:

```
class Solution {
public int[][] resultGrid(int[][] image, int threshold) {

}
}
```

Python3:

```
class Solution:
def resultGrid(self, image: List[List[int]], threshold: int) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def resultGrid(self, image, threshold):
"""
:type image: List[List[int]]
:type threshold: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} image
* @param {number} threshold
* @return {number[][]}
*/
var resultGrid = function(image, threshold) {

};
```

TypeScript:

```
function resultGrid(image: number[][], threshold: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ResultGrid(int[][] image, int threshold) {

}
}
```

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** resultGrid(int** image, int imageSize, int* imageColSize, int
threshold, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func resultGrid(image [][]int, threshold int) [][]int {

}
```

Kotlin:

```
class Solution {
fun resultGrid(image: Array<IntArray>, threshold: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func resultGrid(_ image: [[Int]], _ threshold: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn result_grid(image: Vec<Vec<i32>>, threshold: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} image
# @param {Integer} threshold
# @return {Integer[][]}
def result_grid(image, threshold)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $image
* @param Integer $threshold
* @return Integer[][]
*/
function resultGrid($image, $threshold) {

}
}
```

Dart:

```
class Solution {
List<List<int>> resultGrid(List<List<int>> image, int threshold) {

}
}
```

Scala:

```
object Solution {
def resultGrid(image: Array[Array[Int]], threshold: Int): Array[Array[Int]] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec result_grid(image :: [[integer]], threshold :: integer) :: [[integer]]
def result_grid(image, threshold) do

end
end
```

Erlang:

```
-spec result_grid(Image :: [[integer()]], Threshold :: integer()) ->
[[integer()]].
result_grid(Image, Threshold) ->
.
```

Racket:

```
(define/contract (result-grid image threshold)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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>> resultGrid(vector<vector<int>>& image, int threshold) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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[][] resultGrid(int[][] image, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Grid of Region Average
Difficulty: Medium
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 resultGrid(self, image: List[List[int]], threshold: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def resultGrid(self, image, threshold):
"""
:type image: List[List[int]]
:type threshold: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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[][]} image
* @param {number} threshold
* @return {number[][]}
*/
var resultGrid = function(image, threshold) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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 resultGrid(image: number[][], threshold: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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[][] ResultGrid(int[][] image, int threshold) {

}
}
```

C Solution:

```
/*
* Problem: Find the Grid of Region Average
* Difficulty: Medium
* 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** resultGrid(int** image, int imageSize, int* imageColSize, int
threshold, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find the Grid of Region Average
// Difficulty: Medium
// 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 resultGrid(image [][]int, threshold int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun resultGrid(image: Array<IntArray>, threshold: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func resultGrid(_ image: [[Int]], _ threshold: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Find the Grid of Region Average
// Difficulty: Medium
// 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 result_grid(image: Vec<Vec<i32>>, threshold: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} image
# @param {Integer} threshold
# @return {Integer[][]}
def result_grid(image, threshold)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $image
* @param Integer $threshold
* @return Integer[][]
*/
function resultGrid($image, $threshold) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> resultGrid(List<List<int>> image, int threshold) {

}
}
```

Scala Solution:

```
object Solution {
def resultGrid(image: Array[Array[Int]], threshold: Int): Array[Array[Int]] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec result_grid(image :: [[integer]], threshold :: integer) :: [[integer]]
```

```
def result_grid(image, threshold) do

end
end
```

Erlang Solution:

```
-spec result_grid(Image :: [[integer()]], Threshold :: integer()) ->
[[integer()]].
result_grid(Image, Threshold) ->
.
```

Racket Solution:

```
(define/contract (result-grid image threshold)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```