Problem 2282: Number of People That Can Be Seen in a Grid

Problem Information

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

Problem Description

You are given an

m x n

0-indexed

2D array of positive integers

heights

where

heights[i][j]

is the height of the person standing at position

(i, j)

.

A person standing at position

(row

1

, col

1

)

can see a person standing at position

(row

2

, col

2

)

if:

The person at

(row

2

, col

2

)

is to the right

or

below the person at

(row

1

, col

1

)

. More formally, this means that either

row

1

== row

2

and

col

1

< col

2

or

row

1

< row

2

and

col

1

== col

2

.

Everyone in between them is shorter than

both

of them.

Return

an

m x n

2D array of integers

answer

where

answer[i][j]

is the number of people that the person at position

(i, j)

can see.

Example 1:

heights

3	1	4	2	5

answer

2	1	2	1	0

Input:

heights = [[3,1,4,2,5]]

Output:

[[2,1,2,1,0]]

Explanation:

- The person at (0, 0) can see the people at (0, 1) and (0, 2). Note that he cannot see the person at (0, 4) because the person at (0, 2) is taller than him. - The person at (0, 1) can see the person at (0, 2). - The person at (0, 2) can see the people at (0, 3) and (0, 4). - The person at (0, 3) can see the person at (0, 4). - The person at (0, 4) cannot see anybody.

Example 2:

heights

5	1
3	1
4	1

answer

3	1
2	1
1	0

Input:

heights = [[5,1],[3,1],[4,1]]

Output:

[[3,1],[2,1],[1,0]]

Explanation:

- The person at (0, 0) can see the people at (0, 1), (1, 0) and (2, 0). - The person at (0, 1) can see the person at (1, 1). - The person at (1, 0) can see the people at (1, 1) and (2, 0). - The person at (1, 1) can see the person at (2, 1). - The person at (2, 0) can see the person at (2, 1). - The person at (2, 1) cannot see anybody.

Constraints:

1 <= heights.length <= 400

1 <= heights[i].length <= 400

1 <= heights[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> seePeople(vector<vector<int>>& heights) {

}
};
```

Java:

```
class Solution {
public int[][] seePeople(int[][] heights) {
```

```
}
}
```

Python3:

```
class Solution:
def seePeople(self, heights: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function seePeople(heights: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SeePeople(int[][] heights) {

}
}
```

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** seePeople(int** heights, int heightsSize, int* heightsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func seePeople(heights [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun seePeople(heights: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func seePeople(_ heights: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn see_people(heights: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} heights
# @return {Integer[][]}
def see_people(heights)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $heights
* @return Integer[][]
*/
function seePeople($heights) {

}
}
```

Dart:

```
class Solution {
List<List<int>> seePeople(List<List<int>> heights) {

}
}
```

Scala:

```
object Solution {
def seePeople(heights: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec see_people(heights :: [[integer]]) :: [[integer]]
def see_people(heights) do

end
end
```

Erlang:

```
-spec see_people(Heights :: [[integer()]]) -> [[integer()]].
see_people(Heights) ->
.
```

Racket:

```
(define/contract (see-people heights)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
* Tags: array, stack
*
* 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>> seePeople(vector<vector<int>>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
* Tags: array, stack
*
* 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[][] seePeople(int[][] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of People That Can Be Seen in a Grid
Difficulty: Medium
Tags: array, stack

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 seePeople(self, heights: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
```

```
* Tags: array, stack
*
* 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[][]} heights
* @return {number[][]}
*/
var seePeople = function(heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
* Tags: array, stack
*
* 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 seePeople(heights: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
* Tags: array, stack
*
* 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[][] SeePeople(int[][] heights) {

}
}
```

C Solution:

```
/*
* Problem: Number of People That Can Be Seen in a Grid
* Difficulty: Medium
* Tags: array, stack
*
* 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** seePeople(int** heights, int heightsSize, int* heightsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Number of People That Can Be Seen in a Grid
// Difficulty: Medium
// Tags: array, stack
//
// 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 seePeople(heights [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun seePeople(heights: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func seePeople(_ heights: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Number of People That Can Be Seen in a Grid
// Difficulty: Medium
// Tags: array, stack
//
// 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 see_people(heights: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} heights
# @return {Integer[][]}
def see_people(heights)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $heights
* @return Integer[][]
*/
function seePeople($heights) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> seePeople(List<List<int>> heights) {

}
}
```

Scala Solution:

```
object Solution {
def seePeople(heights: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec see_people(heights :: [[integer]]) :: [[integer]]
def see_people(heights) do

end
end
```

Erlang Solution:

```
-spec see_people(Heights :: [[integer()]]) -> [[integer()]].
see_people(Heights) ->
.
```

Racket Solution:

```
(define/contract (see-people heights)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```