Problem 417: Pacific Atlantic Water Flow

Problem Information

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

Problem Description

There is an

m x n

rectangular island that borders both the

Pacific Ocean

and

Atlantic Ocean

. The

Pacific Ocean

touches the island's left and top edges, and the

Atlantic Ocean

touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an

m x n

integer matrix

heights

where

heights[r][c]

represents the

height above sea level

of the cell at coordinate

(r, c)

.

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is

less than or equal to

the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return

a

2D list

of grid coordinates

result

where

result[i] = [r

i

, c

i

]

denotes that rain water can flow from cell

(r

i

, c

i

)

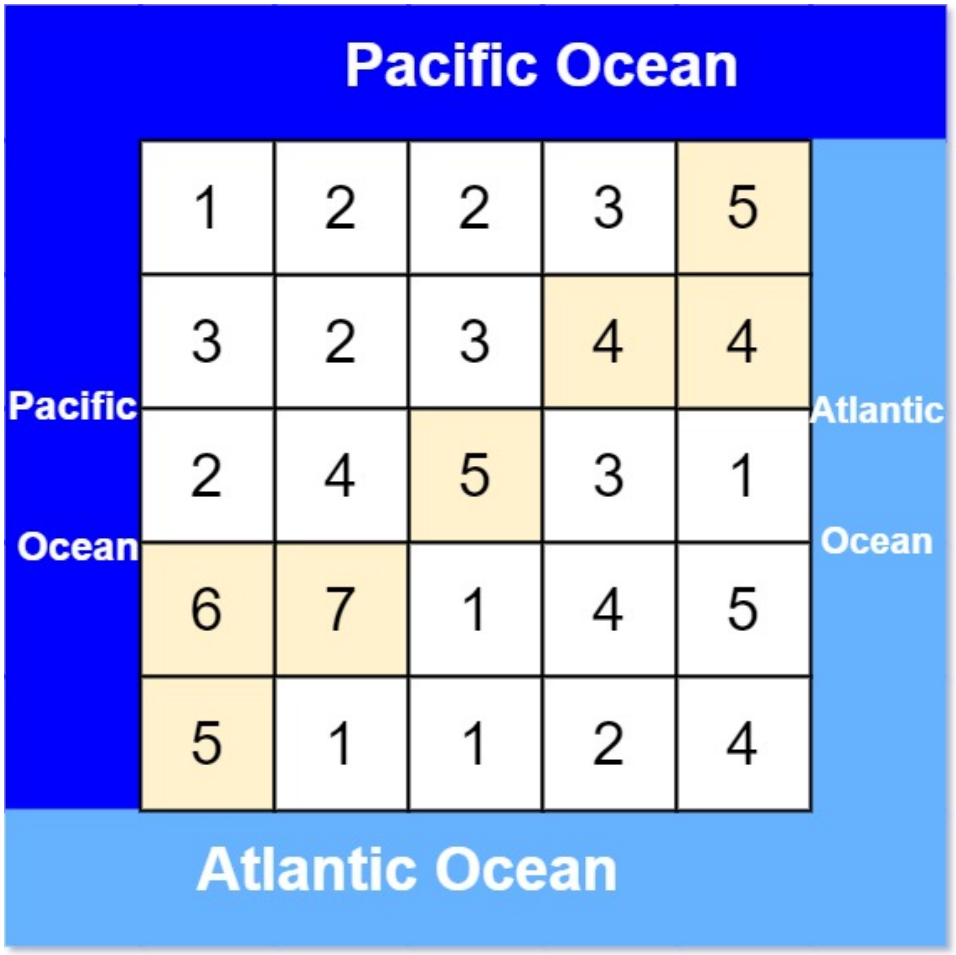
to

both

the Pacific and Atlantic oceans

.

Example 1:



Input:

heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]

Output:

[[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]

Explanation:

The following cells can flow to the Pacific and Atlantic oceans, as shown below: [0,4]: [0,4] -> Pacific Ocean [0,4] -> Atlantic Ocean [1,3]: [1,3] -> [0,3] -> Pacific Ocean [1,3] -> [1,4] -> Atlantic Ocean [1,4]: [1,4] -> [1,3] -> [0,3] -> Pacific Ocean [1,4] -> Atlantic Ocean [2,2]: [2,2] -> [1,2] -> [0,2] -> Pacific Ocean [2,2] -> [2,3] -> [2,4] -> Atlantic Ocean [3,0]: [3,0] -> Pacific Ocean [3,0] -> [4,0] -> Atlantic Ocean [3,1]: [3,1] -> [3,0] -> Pacific Ocean [3,1] -> [4,1] -> Atlantic Ocean [4,0]: [4,0] -> Pacific Ocean [4,0] -> Atlantic Ocean Note that there are other possible paths for these cells to flow to the Pacific and Atlantic oceans.

Example 2:

Input:

heights = [[1]]

Output:

[[0,0]]

Explanation:

The water can flow from the only cell to the Pacific and Atlantic oceans.

Constraints:

m == heights.length

n == heights[r].length

1 <= m, n <= 200

0 <= heights[r][c] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> pacificAtlantic(int[][] heights) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> PacificAtlantic(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** pacificAtlantic(int** heights, int heightsSize, int* heightsColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun pacificAtlantic(heights: Array<IntArray>): List<List<Int>> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (pacific-atlantic heights)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pacific Atlantic Water Flow
* Difficulty: Medium
* Tags: array, search
*
* 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>> pacificAtlantic(vector<vector<int>>& heights) {

}
};
```

Java Solution:

```
/**
* Problem: Pacific Atlantic Water Flow
```

```
* Difficulty: Medium
* Tags: array, search
*
* 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 List<List<Integer>> pacificAtlantic(int[][] heights) {

}
}
```

Python3 Solution:

```
"""
Problem: Pacific Atlantic Water Flow
Difficulty: Medium
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Pacific Atlantic Water Flow
* Difficulty: Medium
* Tags: array, search
*
* 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 pacificAtlantic = function(heights) {

};
```

TypeScript Solution:

```
/**
* Problem: Pacific Atlantic Water Flow
* Difficulty: Medium
* Tags: array, search
*
* 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 pacificAtlantic(heights: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Pacific Atlantic Water Flow
* Difficulty: Medium
* Tags: array, search
*
* 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 IList<IList<int>> PacificAtlantic(int[][] heights) {

}
}
```

C Solution:

```
/*
* Problem: Pacific Atlantic Water Flow
* Difficulty: Medium
* Tags: array, search
*
* 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** pacificAtlantic(int** heights, int heightsSize, int* heightsColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Pacific Atlantic Water Flow
// Difficulty: Medium
// Tags: array, search
//
// 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 pacificAtlantic(heights [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun pacificAtlantic(heights: Array<IntArray>): List<List<Int>> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Pacific Atlantic Water Flow
// Difficulty: Medium
// Tags: array, search
//
// 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 pacific_atlantic(heights: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (pacific-atlantic heights)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```