Problem 1765: Map of Highest Peak

Problem Information

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

Problem Description

You are given an integer matrix

isWater

of size

m x n

that represents a map of

land

and

water

cells.

If

isWater[i][j] == 0

, cell

(i, j)

is a

land

cell.

If

isWater[i][j] == 1

, cell

(i, j)

is a

water

cell.

You must assign each cell a height in a way that follows these rules:

The height of each cell must be non-negative.

If the cell is a

water

cell, its height must be

0

.

Any two adjacent cells must have an absolute height difference of

at most

1

. A cell is adjacent to another cell if the former is directly north, east, south, or west of the latter (i.e., their sides are touching).

Find an assignment of heights such that the maximum height in the matrix is

maximized

.

Return

an integer matrix

height

of size

m x n

where

height[i][j]

is cell

(i, j)

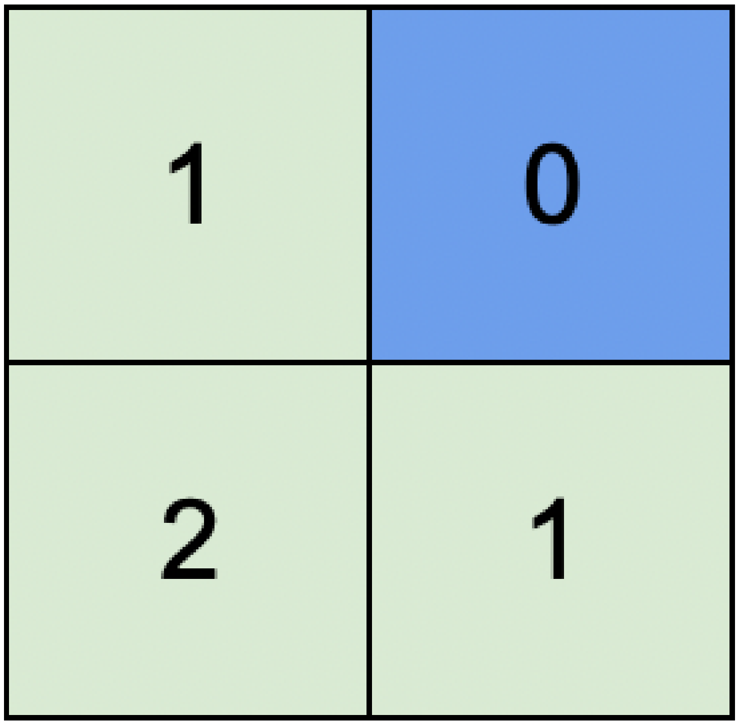
's height. If there are multiple solutions, return

any

of them

.

Example 1:



Input:

isWater = [[0,1],[0,0]]

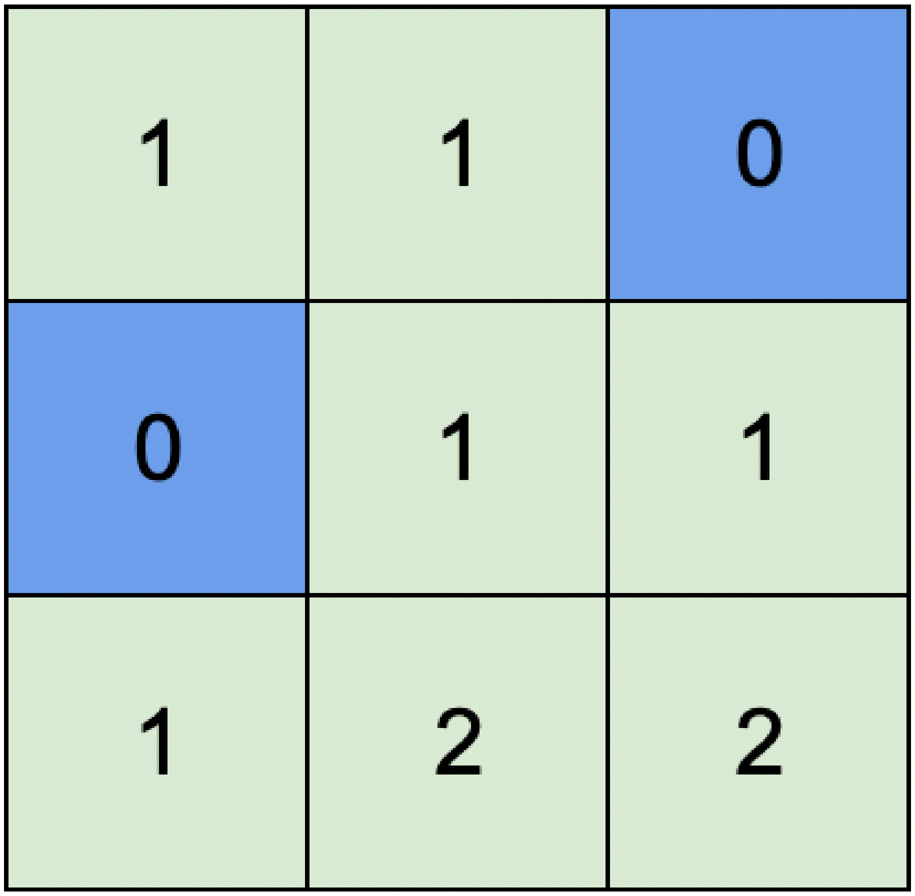
Output:

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

Explanation:

The image shows the assigned heights of each cell. The blue cell is the water cell, and the green cells are the land cells.

Example 2:



Input:

isWater = [[0,0,1],[1,0,0],[0,0,0]]

Output:

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

Explanation:

A height of 2 is the maximum possible height of any assignment. Any height assignment that has a maximum height of 2 while still meeting the rules will also be accepted.

Constraints:

m == isWater.length

n == isWater[i].length

1 <= m, n <= 1000

isWater[i][j]

is

0

or

1

.

There is at least

one

water cell.

Note:

This question is the same as 542:

https://leetcode.com/problems/01-matrix/

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> highestPeak(vector<vector<int>>& isWater) {

}
};
```

Java:

```
class Solution {
public int[][] highestPeak(int[][] isWater) {

}
}
```

Python3:

```
class Solution:
def highestPeak(self, isWater: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function highestPeak(isWater: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] HighestPeak(int[][] isWater) {

}
}
```

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** highestPeak(int** isWater, int isWaterSize, int* isWaterColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func highestPeak(isWater [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun highestPeak(isWater: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func highestPeak(_ isWater: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn highest_peak(is_water: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} is_water
# @return {Integer[][]}
def highest_peak(is_water)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $isWater
* @return Integer[][]
*/
function highestPeak($isWater) {

}
}
```

Dart:

```
class Solution {
List<List<int>> highestPeak(List<List<int>> isWater) {

}
}
```

Scala:

```
object Solution {
def highestPeak(isWater: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec highest_peak(is_water :: [[integer]]) :: [[integer]]
def highest_peak(is_water) do
```

```
end
end
```

Erlang:

```
-spec highest_peak(IsWater :: [[integer()]]) -> [[integer()]].
highest_peak(IsWater) ->
.
```

Racket:

```
(define/contract (highest-peak isWater)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Map of Highest Peak
* 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>> highestPeak(vector<vector<int>>& isWater) {

}
};
```

Java Solution:

```
/**
* Problem: Map of Highest Peak
* 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 int[][] highestPeak(int[][] isWater) {

}
}
```

Python3 Solution:

```
"""
Problem: Map of Highest Peak
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 highestPeak(self, isWater: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Map of Highest Peak
* 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[][]} isWater
* @return {number[][]}
*/
var highestPeak = function(isWater) {

};
```

TypeScript Solution:

```
/**
* Problem: Map of Highest Peak
* 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 highestPeak(isWater: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Map of Highest Peak
* 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 int[][] HighestPeak(int[][] isWater) {

}
}
```

C Solution:

```
/*
* Problem: Map of Highest Peak
* 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** highestPeak(int** isWater, int isWaterSize, int* isWaterColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Map of Highest Peak
// 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 highestPeak(isWater [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun highestPeak(isWater: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func highestPeak(_ isWater: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Map of Highest Peak
// 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 highest_peak(is_water: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} is_water
# @return {Integer[][]}
def highest_peak(is_water)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $isWater
* @return Integer[][]
*/
function highestPeak($isWater) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> highestPeak(List<List<int>> isWater) {

}
}
```

Scala Solution:

```
object Solution {
def highestPeak(isWater: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec highest_peak(is_water :: [[integer]]) :: [[integer]]
def highest_peak(is_water) do

end
```

```

end
```

Erlang Solution:

```
-spec highest_peak(IsWater :: [[integer()]]) -> [[integer()]].
highest_peak(IsWater) ->
.
```

Racket Solution:

```
(define/contract (highest-peak isWater)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```