

Problem 305: Number of Islands II

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an empty 2D binary grid

grid

of size

$m \times n$

. The grid represents a map where

0

's represent water and

1

's represent land. Initially, all the cells of

grid

are water cells (i.e., all the cells are

0

's).

We may perform an add land operation which turns the water at position into a land. You are given an array

positions

where

positions[i] = [r

i

, c

i

]

is the position

(r

i

, c

i

)

at which we should operate the

i

th

operation.

Return

an array of integers

answer

where

answer[i]

is the number of islands after turning the cell

(r

i

, c

i

)

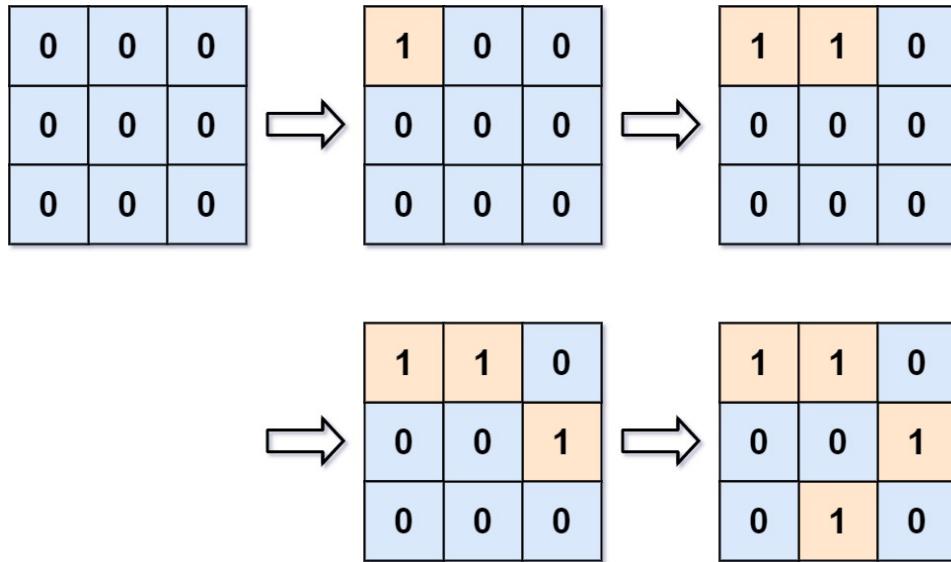
into a land

An

island

is surrounded by water and is formed by connecting adjacent lands horizontally or vertically.
You may assume all four edges of the grid are all surrounded by water.

Example 1:



Input:

$m = 3, n = 3, \text{positions} = [[0,0],[0,1],[1,2],[2,1]]$

Output:

[1,1,2,3]

Explanation:

Initially, the 2d grid is filled with water. - Operation #1: `addLand(0, 0)` turns the water at `grid[0][0]` into a land. We have 1 island. - Operation #2: `addLand(0, 1)` turns the water at `grid[0][1]` into a land. We still have 1 island. - Operation #3: `addLand(1, 2)` turns the water at `grid[1][2]` into a land. We have 2 islands. - Operation #4: `addLand(2, 1)` turns the water at `grid[2][1]` into a land. We have 3 islands.

Example 2:

Input:

$m = 1, n = 1, \text{positions} = [[0,0]]$

Output:

[1]

Constraints:

$1 \leq m, n, \text{positions.length} \leq 10$

4

$1 \leq m * n \leq 10$

4

$\text{positions}[i].length == 2$

$0 \leq r$

i

$< m$

$0 \leq c$

i

$< n$

Follow up:

Could you solve it in time complexity

$O(k \log(mn))$

, where

$k == \text{positions.length}$

?

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> numIslands2(int m, int n, vector<vector<int>>& positions) {  
  
}  
};
```

Java:

```
class Solution {  
public List<Integer> numIslands2(int m, int n, int[][] positions) {  
  
}  
}
```

Python3:

```
class Solution:  
def numIslands2(self, m: int, n: int, positions: List[List[int]]) ->  
List[int]:
```

Python:

```
class Solution(object):  
def numIslands2(self, m, n, positions):  
"""  
:type m: int  
:type n: int  
:type positions: List[List[int]]  
:rtype: List[int]  
"""
```

JavaScript:

```
/**  
 * @param {number} m  
 * @param {number} n  
 * @param {number[][]} positions  
 * @return {number[]}   
 */  
var numIslands2 = function(m, n, positions) {
```

```
};
```

TypeScript:

```
function numIslands2(m: number, n: number, positions: number[][][]): number[] {  
    ...  
}
```

C#:

```
public class Solution {  
    public IList<int> NumIslands2(int m, int n, int[][] positions) {  
        ...  
    }  
}
```

C:

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* numIslands2(int m, int n, int** positions, int positionsSize, int*  
positionsColSize, int* returnSize) {  
    ...  
}
```

Go:

```
func numIslands2(m int, n int, positions [][]int) []int {  
    ...  
}
```

Kotlin:

```
class Solution {  
    fun numIslands2(m: Int, n: Int, positions: Array<IntArray>): List<Int> {  
        ...  
    }  
}
```

Swift:

```
class Solution {  
    func numIslands2(_ m: Int, _ n: Int, _ positions: [[Int]]) -> [Int] {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn num_islands2(m: i32, n: i32, positions: Vec<Vec<i32>>) -> Vec<i32> {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer} m  
# @param {Integer} n  
# @param {Integer[][]} positions  
# @return {Integer[]}  
def num_islands2(m, n, positions)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $m  
     * @param Integer $n  
     * @param Integer[][] $positions  
     * @return Integer[]  
     */  
    function numIslands2($m, $n, $positions) {  
  
    }  
}
```

Dart:

```
class Solution {  
    List<int> numIslands2(int m, int n, List<List<int>> positions) {  
    }
```

```
}
```

```
}
```

Scala:

```
object Solution {  
    def numIslands2(m: Int, n: Int, positions: Array[Array[Int]]): List[Int] = {  
  
    }  
    }  
}
```

Elixir:

```
defmodule Solution do  
  @spec num_islands2(m :: integer, n :: integer, positions :: [[integer]]) ::  
  [integer]  
  def num_islands2(m, n, positions) do  
  
  end  
  end
```

Erlang:

```
-spec num_islands2(M :: integer(), N :: integer(), Positions ::  
  [[integer()]]) -> [integer()].  
num_islands2(M, N, Positions) ->  
.
```

Racket:

```
(define/contract (num-islands2 m n positions)  
  (-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof  
  exact-integer?)))  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Number of Islands II
 * Difficulty: Hard
 * Tags: array, graph, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
vector<int> numIslands2(int m, int n, vector<vector<int>>& positions) {

}
};


```

Java Solution:

```

/**
 * Problem: Number of Islands II
 * Difficulty: Hard
 * Tags: array, graph, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public List<Integer> numIslands2(int m, int n, int[][] positions) {

}
}


```

Python3 Solution:

```

"""
Problem: Number of Islands II
Difficulty: Hard
Tags: array, graph, hash

```

```

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map

"""

class Solution:

def numIslands2(self, m: int, n: int, positions: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def numIslands2(self, m, n, positions):
"""
:type m: int
:type n: int
:type positions: List[List[int]]
:rtype: List[int]
"""


```

JavaScript Solution:

```

/**
 * Problem: Number of Islands II
 * Difficulty: Hard
 * Tags: array, graph, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

/**
 * @param {number} m
 * @param {number} n
 * @param {number[][]} positions
 * @return {number[]}
 */
var numIslands2 = function(m, n, positions) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Number of Islands II  
 * Difficulty: Hard  
 * Tags: array, graph, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
function numIslands2(m: number, n: number, positions: number[][][]): number[] {  
}  
};
```

C# Solution:

```
/*  
 * Problem: Number of Islands II  
 * Difficulty: Hard  
 * Tags: array, graph, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
public class Solution {  
    public IList<int> NumIslands2(int m, int n, int[][] positions) {  
        return null;  
    }  
}
```

C Solution:

```
/*  
 * Problem: Number of Islands II
```

```

* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

```

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* numIslands2(int m, int n, int** positions, int positionsSize, int*
positionsColSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Number of Islands II
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func numIslands2(m int, n int, positions [][]int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun numIslands2(m: Int, n: Int, positions: Array<IntArray>): List<Int> {
        }
    }
}

```

Swift Solution:

```

class Solution {

func numIslands2(_ m: Int, _ n: Int, _ positions: [[Int]]) -> [Int] {

}

}

```

Rust Solution:

```

// Problem: Number of Islands II
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn num_islands2(m: i32, n: i32, positions: Vec<Vec<i32>>) -> Vec<i32> {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer} m
# @param {Integer} n
# @param {Integer[][]} positions
# @return {Integer[]}
def num_islands2(m, n, positions)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer $m
     * @param Integer $n
     * @param Integer[][] $positions
     * @return Integer[]
     */
}

```

```
function numIslands2($m, $n, $positions) {  
}  
}  
}
```

Dart Solution:

```
class Solution {  
List<int> numIslands2(int m, int n, List<List<int>> positions) {  
}  
}  
}
```

Scala Solution:

```
object Solution {  
def numIslands2(m: Int, n: Int, positions: Array[Array[Int]]): List[Int] = {  
}  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec num_islands2(m :: integer, n :: integer, positions :: [[integer]]) ::  
[integer]  
def num_islands2(m, n, positions) do  
  
end  
end
```

Erlang Solution:

```
-spec num_islands2(M :: integer(), N :: integer(), Positions ::  
[[integer()]]) -> [integer()].  
num_islands2(M, N, Positions) ->  
.
```

Racket Solution:

```
(define/contract (num-islands2 m n positions)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
exact-integer?)))
)
```