Problem 803: Bricks Falling When Hit

Problem Information

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

Problem Description

You are given an

m x n

binary

grid

, where each

1

represents a brick and

0

represents an empty space. A brick is

stable

if:

It is directly connected to the top of the grid, or

At least one other brick in its four adjacent cells is

stable

.

You are also given an array

hits

, which is a sequence of erasures we want to apply. Each time we want to erase the brick at the location

hits[i] = (row

i

, col

i

)

. The brick on that location (if it exists) will disappear. Some other bricks may no longer be stable because of that erasure and will

fall

. Once a brick falls, it is

immediately

erased from the

grid

(i.e., it does not land on other stable bricks).

Return

an array

result

, where each

result[i]

is the number of bricks that will

fall

after the

i

th

erasure is applied.

Note

that an erasure may refer to a location with no brick, and if it does, no bricks drop.

Example 1:

Input:

grid = [[1,0,0,0],[1,1,1,0]], hits = [[1,0]]

Output:

[2]

Explanation:

Starting with the grid: [[1,0,0,0], [

1

,1,1,0]] We erase the underlined brick at (1,0), resulting in the grid: [[1,0,0,0], [0,

1

,

1

,0]] The two underlined bricks are no longer stable as they are no longer connected to the top nor adjacent to another stable brick, so they will fall. The resulting grid is: [[1,0,0,0], [0,0,0,0]] Hence the result is [2].

Example 2:

Input:

grid = [[1,0,0,0],[1,1,0,0]], hits = [[1,1],[1,0]]

Output:

[0,0]

Explanation:

Starting with the grid: [[1,0,0,0], [1,

1

,0,0]] We erase the underlined brick at (1,1), resulting in the grid: [[1,0,0,0], [1,0,0,0]] All remaining bricks are still stable, so no bricks fall. The grid remains the same: [[1,0,0,0], [

1

,0,0,0]] Next, we erase the underlined brick at (1,0), resulting in the grid: [[1,0,0,0], [0,0,0,0]] Once again, all remaining bricks are still stable, so no bricks fall. Hence the result is [0,0].

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 200

grid[i][j]

is

0

or

1

.

1 <= hits.length <= 4 * 10

4

hits[i].length == 2

0 <= x

i

<= m - 1

0 <= y

i

<= n - 1

All

(x

i

, y

i

)

are unique.

Code Snippets

C++:

```
class Solution {
public:
vector<int> hitBricks(vector<vector<int>>& grid, vector<vector<int>>& hits) {

}
};
```

Java:

```
class Solution {
public int[] hitBricks(int[][] grid, int[][] hits) {

}
}
```

Python3:

```
class Solution:
def hitBricks(self, grid: List[List[int]], hits: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def hitBricks(self, grid, hits):
"""
```

```
:type grid: List[List[int]]
:type hits: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number[][]} hits
* @return {number[]}
*/
var hitBricks = function(grid, hits) {

};
```

TypeScript:

```
function hitBricks(grid: number[][], hits: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] HitBricks(int[][] grid, int[][] hits) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* hitBricks(int** grid, int gridSize, int* gridColSize, int** hits, int
hitsSize, int* hitsColSize, int* returnSize) {

}
```

Go:

```
func hitBricks(grid [][]int, hits [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun hitBricks(grid: Array<IntArray>, hits: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func hitBricks(_ grid: [[Int]], _ hits: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn hit_bricks(grid: Vec<Vec<i32>>, hits: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer[][]} hits
# @return {Integer[]}
def hit_bricks(grid, hits)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
```

```
* @param Integer[][] $hits
* @return Integer[]
*/
function hitBricks($grid, $hits) {

}
}
```

Dart:

```
class Solution {
List<int> hitBricks(List<List<int>> grid, List<List<int>> hits) {

}
}
```

Scala:

```
object Solution {
def hitBricks(grid: Array[Array[Int]], hits: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec hit_bricks(grid :: [[integer]], hits :: [[integer]]) :: [integer]
def hit_bricks(grid, hits) do

end
end
```

Erlang:

```
-spec hit_bricks(Grid :: [[integer()]], Hits :: [[integer()]]) ->
[integer()].
hit_bricks(Grid, Hits) ->
.
```

Racket:

```
(define/contract (hit-bricks grid hits)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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<int> hitBricks(vector<vector<int>>& grid, vector<vector<int>>& hits) {

}
};
```

Java Solution:

```
/**
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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[] hitBricks(int[][] grid, int[][] hits) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Bricks Falling When Hit
Difficulty: Hard
Tags: array, graph

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 hitBricks(self, grid: List[List[int]], hits: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def hitBricks(self, grid, hits):
"""
:type grid: List[List[int]]
:type hits: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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[][]} grid
* @param {number[][]} hits
* @return {number[]}
*/
var hitBricks = function(grid, hits) {

};
```

TypeScript Solution:

```
/**
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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 hitBricks(grid: number[][], hits: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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[] HitBricks(int[][] grid, int[][] hits) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Bricks Falling When Hit
* Difficulty: Hard
* Tags: array, graph
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* hitBricks(int** grid, int gridSize, int* gridColSize, int** hits, int
hitsSize, int* hitsColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Bricks Falling When Hit
// Difficulty: Hard
// Tags: array, graph
//
// 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 hitBricks(grid [][]int, hits [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun hitBricks(grid: Array<IntArray>, hits: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func hitBricks(_ grid: [[Int]], _ hits: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Bricks Falling When Hit
// Difficulty: Hard
// Tags: array, graph
//
// 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 hit_bricks(grid: Vec<Vec<i32>>, hits: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer[][]} hits
# @return {Integer[]}
def hit_bricks(grid, hits)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[][] $hits
* @return Integer[]
*/
function hitBricks($grid, $hits) {

}
}
```

Dart Solution:

```
class Solution {
List<int> hitBricks(List<List<int>> grid, List<List<int>> hits) {

}
}
```

Scala Solution:

```
object Solution {
def hitBricks(grid: Array[Array[Int]], hits: Array[Array[Int]]): Array[Int] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec hit_bricks(grid :: [[integer]], hits :: [[integer]]) :: [integer]
def hit_bricks(grid, hits) do

end
end
```

Erlang Solution:

```
-spec hit_bricks(Grid :: [[integer()]], Hits :: [[integer()]]) ->
[integer()].
```

```
hit_bricks(Grid, Hits) ->
.
```

Racket Solution:

```
(define/contract (hit-bricks grid hits)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```