Problem 407: Trapping Rain Water II

Problem Information

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

Problem Description

Given an

m x n

integer matrix

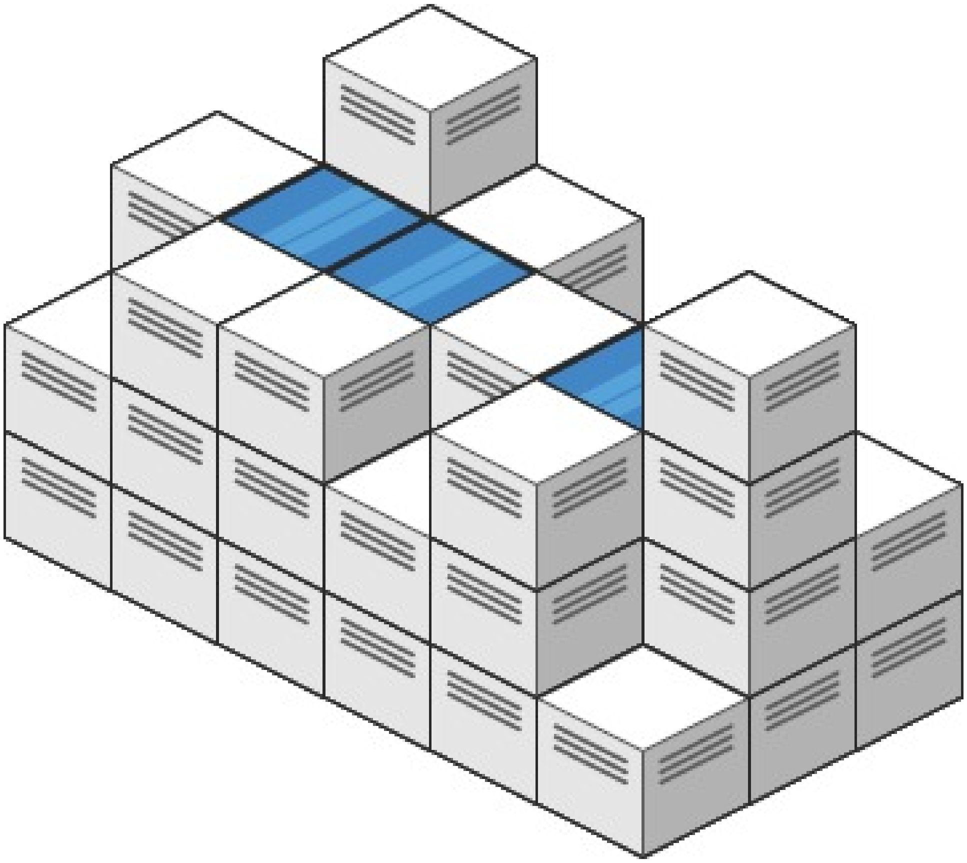
heightMap

representing the height of each unit cell in a 2D elevation map, return

the volume of water it can trap after raining

.

Example 1:



Input:

heightMap = [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]]

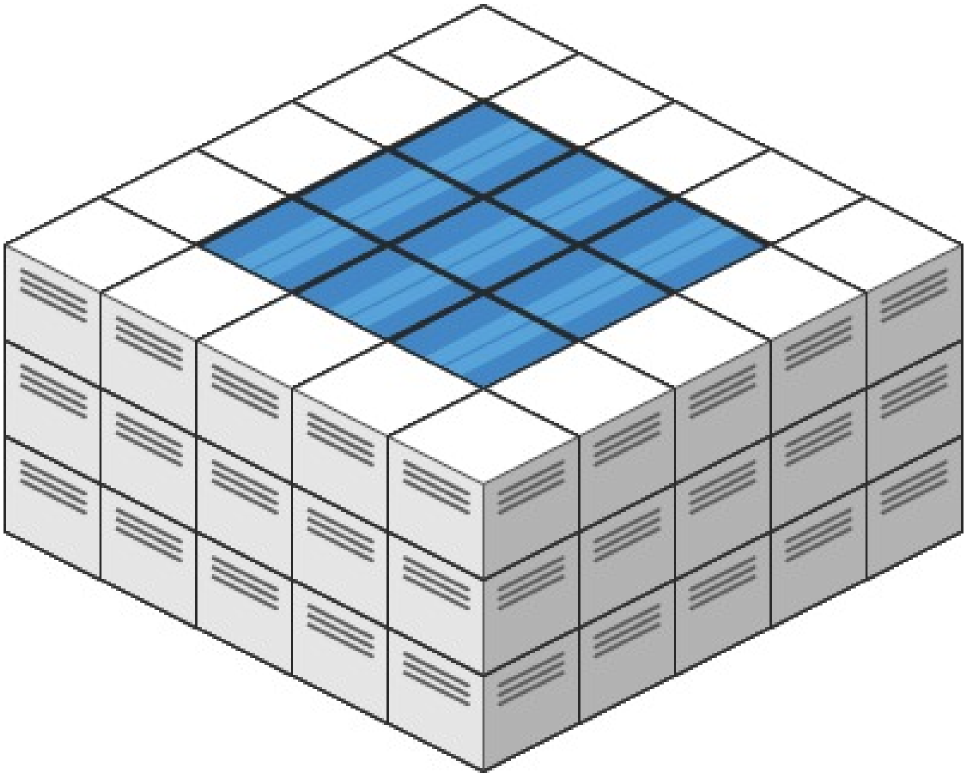
Output:

4

Explanation:

After the rain, water is trapped between the blocks. We have two small ponds 1 and 3 units trapped. The total volume of water trapped is 4.

Example 2:



Input:

heightMap = [[3,3,3,3,3],[3,2,2,2,3],[3,2,1,2,3],[3,2,2,2,3],[3,3,3,3,3]]

Output:

10

Constraints:

m == heightMap.length

n == heightMap[i].length

1 <= m, n <= 200

0 <= heightMap[i][j] <= 2 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int trapRainWater(vector<vector<int>>& heightMap) {

}
};
```

Java:

```
class Solution {
public int trapRainWater(int[][] heightMap) {

}
}
```

Python3:

```
class Solution:
def trapRainWater(self, heightMap: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function trapRainWater(heightMap: number[][]): number {

};
```

C#:

```
public class Solution {
public int TrapRainWater(int[][] heightMap) {

}
}
```

C:

```
int trapRainWater(int** heightMap, int heightMapSize, int* heightMapColSize)
{

}
```

Go:

```
func trapRainWater(heightMap [][]int) int {

}
```

Kotlin:

```
class Solution {
fun trapRainWater(heightMap: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func trapRainWater(_ heightMap: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn trap_rain_water(height_map: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} height_map
# @return {Integer}
def trap_rain_water(height_map)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $heightMap
* @return Integer
*/
function trapRainWater($heightMap) {

}
}
```

Dart:

```
class Solution {
int trapRainWater(List<List<int>> heightMap) {

}
}
```

Scala:

```
object Solution {
def trapRainWater(heightMap: Array[Array[Int]]): Int = {

}


}
```

Elixir:

```
defmodule Solution do
@spec trap_rain_water(height_map :: [[integer]]) :: integer
def trap_rain_water(height_map) do

end
end
```

Erlang:

```
-spec trap_rain_water(HeightMap :: [[integer()]]) -> integer().
trap_rain_water(HeightMap) ->
.
```

Racket:

```
(define/contract (trap-rain-water heightMap)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 trapRainWater(vector<vector<int>>& heightMap) {
}
};
```

Java Solution:

```
/**
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 trapRainWater(int[][] heightMap) {

}
}
```

Python3 Solution:

```
"""
Problem: Trapping Rain Water II
Difficulty: Hard
Tags: array, search, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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[][]} heightMap
* @return {number}
*/
var trapRainWater = function(heightMap) {

};
```

TypeScript Solution:

```
/**
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 trapRainWater(heightMap: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 TrapRainWater(int[][] heightMap) {

}
}
```

C Solution:

```
/*
* Problem: Trapping Rain Water II
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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
*/

int trapRainWater(int** heightMap, int heightMapSize, int* heightMapColSize)
{

}
```

Go Solution:

```
// Problem: Trapping Rain Water II
// Difficulty: Hard
// Tags: array, search, queue, heap
//
// 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 trapRainWater(heightMap [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun trapRainWater(heightMap: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func trapRainWater(_ heightMap: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Trapping Rain Water II
// Difficulty: Hard
// Tags: array, search, queue, heap
//
// 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 trap_rain_water(height_map: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} height_map
# @return {Integer}
def trap_rain_water(height_map)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $heightMap
* @return Integer
*/
function trapRainWater($heightMap) {

}
}
```

Dart Solution:

```
class Solution {
int trapRainWater(List<List<int>> heightMap) {

}
}
```

Scala Solution:

```
object Solution {
def trapRainWater(heightMap: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec trap_rain_water(height_map :: [[integer]]) :: integer
def trap_rain_water(height_map) do

end
```

```
end
```

Erlang Solution:

```
-spec trap_rain_water(HeightMap :: [[integer()]]) -> integer().
trap_rain_water(HeightMap) ->
  .
```

Racket Solution:

```
(define/contract (trap-rain-water heightMap)
  (-> (listof (listof exact-integer?)) exact-integer?)
  )
```