Problem 42: Trapping Rain Water

Problem Information

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

Problem Description

Given

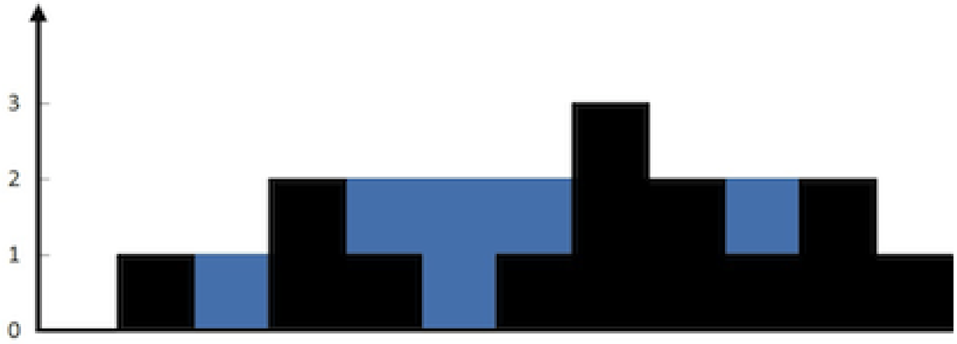
n

non-negative integers representing an elevation map where the width of each bar is

1

, compute how much water it can trap after raining.

Example 1:



Input:

height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output:

6

Explanation:

The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input:

height = [4,2,0,3,2,5]

Output:

9

Constraints:

n == height.length

1 <= n <= 2 * 10

4

0 <= height[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int trap(vector<int>& height) {

}
};
```

Java:

```
class Solution {
public int trap(int[] height) {

}
}
```

Python3:

```
class Solution:
def trap(self, height: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function trap(height: number[]): number {

};
```

C#:

```
public class Solution {
public int Trap(int[] height) {

}
}
```

C:

```
int trap(int* height, int heightSize) {

}
```

Go:

```
func trap(height []int) int {

}
```

Kotlin:

```
class Solution {
fun trap(height: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func trap(_ height: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn trap(height: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} height
# @return {Integer}
def trap(height)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $height
* @return Integer
*/
function trap($height) {

}
}
```

Dart:

```
class Solution {
int trap(List<int> height) {

}
}
```

Scala:

```
object Solution {
def trap(height: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec trap(height :: [integer]) :: integer
def trap(height) do

end
end
```

Erlang:

```erlang
-spec trap(Height :: [integer()]) -> integer().
trap(Height) ->
.
```

Racket:

```racket
(define/contract (trap height)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```cpp
/*
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int trap(vector<int>& height) {

}
};
```

Java Solution:

```java
/**
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int trap(int[] height) {

}
}
```

Python3 Solution:

```
"""
Problem: Trapping Rain Water
Difficulty: Hard
Tags: array, dp, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def trap(self, height: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} height
* @return {number}
*/
var trap = function(height) {

};
```

TypeScript Solution:

```
/**
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function trap(height: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

public class Solution {
public int Trap(int[] height) {

}
}
```

C Solution:

```
/*
* Problem: Trapping Rain Water
* Difficulty: Hard
* Tags: array, dp, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int trap(int* height, int heightSize) {

}
```

Go Solution:

```
// Problem: Trapping Rain Water
// Difficulty: Hard
// Tags: array, dp, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func trap(height []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun trap(height: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func trap(_ height: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Trapping Rain Water
// Difficulty: Hard
// Tags: array, dp, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn trap(height: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} height
# @return {Integer}
def trap(height)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $height
* @return Integer
*/
function trap($height) {

}
}
```

Dart Solution:

```
class Solution {
int trap(List<int> height) {

}
}
```

Scala Solution:

```
object Solution {
def trap(height: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec trap(height :: [integer]) :: integer
def trap(height) do

end
end
```

Erlang Solution:

```
-spec trap(Height :: [integer()]) -> integer().
trap(Height) ->
.
```

Racket Solution:

```
(define/contract (trap height)
(-> (listof exact-integer?) exact-integer?)
)
```