Codility_

CodeCheck Report: training4RJ33N-TPP

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
FloodDepth Python ⚠	4 min	100%

Total score

Tasks Details

1. **FloodDepth**

Medium

Find the maximum depth of water in mountains after a huge rainfall.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are helping a geologist friend investigate an area with mountain lakes. A recent heavy rainfall has flooded these lakes and their water levels have reached the highest possible point. Your friend is interested to know the maximum depth in the deepest part of these lakes.

We simplify the problem in 2-D dimensions. The whole landscape can be divided into small blocks and described by an array A of length N. Each element of A is the altitude of the rock floor of a block (i.e. the height of this block when there is no water at all). After the rainfall, all the low-lying areas (i.e. blocks that have higher blocks

Solution

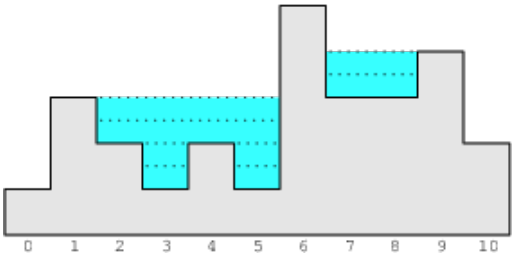
Programming language used:	Python
Total time used:	4 minutes
Effective time used:	4 minutes
Notes:	*not defined yet*

on both sides) are holding as much water as possible. You would like to know the maximum depth of water after this entire area is flooded. You can assume that the altitude outside this area is zero and the outside area can accommodate infinite amount of water.

For example, consider array A such that:

```
A[0] = 1
A[1] = 3
A[2] = 2
A[3] = 1
A[4] = 2
A[5] = 1
A[6] = 5
A[7] = 3
A[8] = 3
A[9] = 4
A[10] = 2
```

The following picture illustrates the landscape after it has flooded:



The gray area is the rock floor described by the array A above and the blue area with dashed lines represents the water filling the low-lying areas with maximum possible volume. Thus, blocks 3 and 5 have a water depth of 2 while blocks 2, 4, 7 and 8 have a water depth of 1. Therefore, the maximum water depth of this area is 2.

Write a function:

```
def solution(A)
```

that, given a non-empty array A consisting of N integers, returns the maximum depth of water.

Given array A shown above, the function should return 2, as explained above.

For the following array:

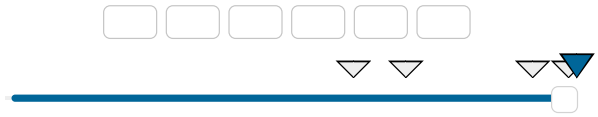
```
A[0] = 5
A[1] = 8
```

the function should return 0, because this landscape cannot hold any water.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..100,000];
- each element of array A is an integer within the range [1..100,000,000].

Task timeline



12:36:27 12:39:52

Code: 12:39:52 UTC, py, final, score: **100** show code in pop-up

```
1  # you can write to stdout for debugging
2  # print("this is a debug message")
3
4  def solution(A):
5      # Implement your solution here
6      n = len(A)
7      left_max = [0] * n
8      right_max = [0] * n
9
10     # Calculate the maximum height on t
11     left_max[0] = A[0]
12     for i in range(1, n):
13         left_max[i] = max(left_max[i -
14
15     # Calculate the maximum height on t
16     right_max[n - 1] = A[n - 1]
17     for i in range(n - 2, -1, -1):
18         right_max[i] = max(right_max[i
19
20     maxDepth = 0
21
22     # Calculate the depth at each posit
23     for i in range(1, n - 1):
24         depth = min(left_max[i - 1], ri
25         if depth > 0:
26             maxDepth = max(maxDepth, de
27
28     return maxDepth
29
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	Example tests	
▶	example1 first example test	✔ OK
▶	example2 second example test	✔ OK
expand all	**Correctness tests**	
▶	boundary	✔ OK

	max-min values and some random tests, N <= 15	
▶	triple_permutation all permutations of three elements [1, 2, 3]	✔ OK
▶	small_functional small functional tests, N <= 15	✔ OK
▶	small_random small random tests, N <= 100	✔ OK
▶	multiple_valleys many small valleys, N <= 100	✔ OK
▶	small_valley one small valley, N <= 100	✔ OK
expand all	**Performance tests**	
▶	medium_tests medium tests, N <= 600	✔ OK
▶	big_random random arrays, N ~= 100,000	✔ OK
▶	increasing_decreasing randomized values which grow when distance from the center decreases, N ~= 100,000	✔ OK
▶	many_multiple_valleys many small valleys, N ~= 100,000	✔ OK
▶	big_valley one big valley, N ~= 100,000	✔ OK
▶	big_jagged one big valley with jagged slopes, N ~= 100,000	✔ OK