Open Feedback Dialog

codility

Training center

Check out Codility training tasks

Demo ticket

Session

ID: demo9GJUDF-VTZ
Time limit: 120 min.

Status: closed

Created on: 2014-03-22 18:44 UTC
Started on: 2014-03-22 18:44 UTC
Finished on: 2014-03-22 18:45 UTC

Tasks in test	Task score
1 | {} ~~Peaks~~	100%

Test score

?

100%

100 out of 100 points

MEDIUM

1. Peaks

Divide an array into the maximum number of same((-))sized blocks, each of which should contain an index P such that A[P - 1] < A[P] > A[P + 1].

score: 100 of 100

Task description

A non-empty zero-indexed array A consisting of N integers is given.
A *peak* is an array element which is larger than its neighbors. More precisely, it is an index P such that 0 < P < N − 1, A[P − 1] < A[P] and A[P] > A[P + 1].
For example, the following array A:

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

A[9] = 4
A[10] = 6
A[11] = 2
```

has exactly three peaks: 3, 5, 10.
We want to divide this array into blocks containing the same number of elements. More precisely, we want to choose a number K that will yield the following blocks:

- A[0], A[1], ..., A[K − 1],
- A[K], A[K + 1], ..., A[2K − 1],
  ...
- A[N − K], A[N − K + 1], ..., A[N − 1].

What's more, every block should contain at least one peak. Notice that extreme elements of the blocks (for example A[K − 1] or A[K]) can also be peaks, but only if they have both neighbors (including one in an adjacent blocks).
The goal is to find the maximum number of blocks into which the array A can be divided.
Array A can be divided into blocks as follows:

- one block (1, 2, 3, 4, 3, 4, 1, 2, 3, 4, 6, 2). This block contains three peaks.

Solution

Programming language used: Python

Total time used: 1 minutes ?

Effective time used: 1 minutes ?

Notes: correct functionality and scalability

Task timeline ?

18:44:42 18:45:14

Code: 18:45:14 UTC, py, final, score: **100.00**

```
def solution(A):
    N = len(A)
    peaks = [-1] * N
    count = 0
    for i in xrange(1, N-1):
        peaks[i] = peaks[i-1]
        if A[i - 1] < A[i] > A[i + 1]:
            peaks[i] = i
            count += 1
    peaks[N-1] = peaks[N-2]

    num = count
    while num > 0:
        if N % num == 0:
            step = N / num
            isvalid = True
            i = 0
            while i < N:
                if peaks[i + step - 1] < i:
                    isvalid = False
                    break
```

- two blocks (1, 2, 3, 4, 3, 4) and (1, 2, 3, 4, 6, 2). Every block has a peak.
- three blocks (1, 2, 3, 4), (3, 4, 1, 2), (3, 4, 6, 2). Every block has a peak. Notice in particular that the first block (1, 2, 3, 4) has a peak at A[3], because A[2] < A[3] > A[4], even though A[4] is in the adjacent block.

However, array A cannot be divided into four blocks, (1, 2, 3), (4, 3, 4), (1, 2, 3) and (4, 6, 2), because the (1, 2, 3) blocks do not contain a peak. Notice in particular that the (4, 3, 4) block contains two peaks: A[3] and A[5].
The maximum number of blocks that array A can be divided into is three.
Write a function:

```
def solution(A)
```

that, given a non-empty zero-indexed array A consisting of N integers, returns the maximum number of blocks into which A can be divided.
If A cannot be divided into some number of blocks, the function should return 0.
For example, given:

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

A[11] = 2
```

the function should return 3, as explained above.
Assume that:

- N is an integer within the range [1..100,000];
- each element of array A is an integer within the range [0..1,000,000,000].

Complexity:

- expected worst-case time complexity is O(N*log(log(N)));
- expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

```
                i += step
            if isvalid:
                return num
        num -= 1
    return num
```

Analysis

?

Detected time complexity:

O(N * log(log(N)))

test	time	result
example example test	0.050 s.	**OK**
extreme_min extreme min test	0.050 s.	**OK**
extreme_without_peaks test without peaks	0.050 s.	**OK**
prime_length test with prime sequence length	0.050 s.	**OK**
anti_bin_search anti bin_search test	0.050 s.	**OK**
simple1 simple test	0.050 s.	**OK**
simple2 second simple test	0.050 s.	**OK**
medium_random chaotic medium sequences, length = ~5,000	0.050 s.	**OK**
medium_anti_slow medium test anti slow solutions	0.070 s.	**OK**
large_random chaotic large sequences, length = ~50,000	0.140 s.	**OK**
large_anti_slow large test anti slow solutions	0.230 s.	**OK**
extreme_max extreme max test	0.250 s.	**OK**

Training center