Codility_

CodeCheck Report: trainingGCMWAW-C6P

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task		Time spent	Score
CountDistinctSlices Python	⚠	5 min	100%

Total score

100%

Tasks Details

		Task Score	Correctness	Performance
Easy	1. CountDistinctSlices Count the number of distinct slices (containing only unique numbers).	100%	100%	100%

Task description

An integer M and a non-empty array A consisting of N non-negative integers are given. All integers in array A are less than or equal to M.

A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a *slice* of array A. The slice consists of the elements A[P], A[P + 1], ..., A[Q]. A *distinct slice* is a slice consisting of only unique numbers. That is, no individual number occurs more than once in the slice.

For example, consider integer M = 6 and array A such that:

```
A[0] = 3
A[1] = 4
A[2] = 5
A[3] = 5
A[4] = 2
```

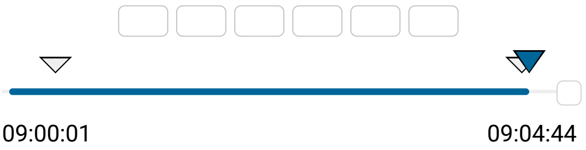
There are exactly nine distinct slices: (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2), (3, 3), (3, 4) and (4, 4).

Solution

Programming language used:	Python	
Total time used:	5 minutes	?
Effective time used:	5 minutes	?
Notes:	*not defined yet*	

Task timeline

?



Code: 09:04:43 UTC, py, show code in pop-up

The goal is to calculate the number of distinct slices.

Write a function:

```
def solution(M, A)
```

that, given an integer M and a non-empty array A consisting of N integers, returns the number of distinct slices.

If the number of distinct slices is greater than 1,000,000,000, the function should return 1,000,000,000.

For example, given integer M = 6 and array A such that:

```
A[0] = 3
A[1] = 4
A[2] = 5
A[3] = 5
A[4] = 2
```

the function should return 9, as explained above.

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

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

final, score: **100**

```
# you can write to stdout for debugging pur
# print("this is a debug message")

def solution(M, A):
    # Implement your solution here
    #pass

    array_length = len(A)
    total_slices = 0
    slice_start = 0
    last_occurrence = [-1] * (M + 1)

    for slice_end in range(array_length):

        if last_occurrence[A[slice_end]] >=
            slice_start = last_occurrence[A

        last_occurrence[A[slice_end]] = sli

        total_slices += slice_end - slice_s


        if total_slices > 1e9:
            return int(1e9)

    return total_slices

```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all **Example tests**

▶ example ✔ **OK**
example test

expand all **Correctness tests**

▶ single ✔ **OK**
single element

▶ double ✔ **OK**
double elements

▶ simple1 ✔ **OK**
first simple test

▶ simple2 ✔ **OK**
second simple test

▶ small_random ✔ **OK**
small random test, length = 100

expand all **Performance tests**

▶ medium_random ✔ **OK**
medium random test, length = 500

▶ large ✔ **OK**
large tests, length = ~100,000

▶ large_range ✔ **OK**

	large range tests, length = ~100,000	
▶	large_random large random tests, length = ~100,000	✔ OK
▶	extreme_the_same all the same elements, length = ~100,000	✔ OK