Codility_

Candidate Report: trainingVMFMYM-ND7

Check out Codility training tasks

Test Name:

Summary Timeline

Tasks summary

Task	Time spent	Score
LeaderSliceInc Go	3 min	100%

Total score

100%

Tasks Details

Hard

1. LeaderSliceInc

Given an array, find all its elements that can become a leader, after increasing by 1 all of the numbers in some segment of a given length.

Task Score	Correctness	Performance
100%	100%	100%

Task description

Integers K, M and a non-empty array A consisting of N integers, not bigger than M, are given.

The leader of the array is a value that occurs in more than half of the elements of the array, and the segment of the array is a sequence of consecutive elements of the array.

You can modify A by choosing exactly one segment of length K and increasing by 1 every element within that segment.

The goal is to find all of the numbers that may become a leader after performing exactly one array modification as described above.

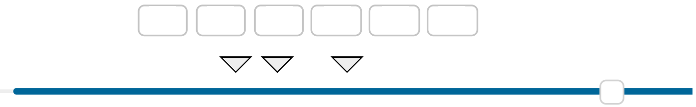
Write a function:

```
    func Solution(K int, M int, A []int) []int
```

Solution

Programming language used:	Go
Total time used:	3 minutes
Effective time used:	3 minutes
Notes:	*not defined yet*

Task timeline



that, given integers K and M and an array A consisting of N integers, returns an array of all numbers that can become a leader, after increasing by 1 every element of exactly one segment of A of length K. The returned array should be sorted in ascending order, and if there is no number that can become a leader, you should return an empty array. Moreover, if there are multiple ways of choosing a segment to turn some number into a leader, then this particular number should appear in an output array only once.

For example, given integers K = 3, M = 5 and the following array A:

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

the function should return [2, 3]. If we choose segment A[1], A[2], A[3] then we get the following array A:

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

and 2 is the leader of this array. If we choose A[3], A[4], A[5] then A will appear as follows:

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

and 3 will be the leader.

And, for example, given integers K = 4, M = 2 and the following array:

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

the function should return [2, 3], because choosing a segment A[0], A[1], A[2], A[3] and A[1], A[2], A[3], A[4] turns 2 and 3 into the leaders, respectively.

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

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

11:37:01 11:39:31

Code: 11:39:31 UTC, go, final, score: **100**

show code in pop-up

```
package solution

import (
    "sort"
)

func computeLeader(
    kChunkCount map[int]int, numCount map[int
    halfCount int, leader int) {
    leaderCount := 0
    leaderCount += kChunkCount[leader-1]
    leaderCount += numCount[leader]
    if leaderCount >= halfCount {
        previousKAns[leader] = true
    } else if _, ok := previousKAns[leader]; o
        delete(previousKAns, leader)
    }
}

func Solution(k int, m int, arr []int) []int
    // fmt.Println(arr)
    n := len(arr)

    numCount := make(map[int]int)
    var num int
    // Process arr entried k..n-1 as we'll sta
    // 0..k-1
    for i := k; i < n; i++ {
        num = arr[i]
        if _, ok := numCount[num]; !ok {
            numCount[num] = 0
        }
        numCount[num]++
    }

    // Process the first k-chunk starting fro
    kChunkCount := make(map[int]int)
    previousKAns := make(map[int]bool)
    // var leaderCount int
    halfCount := n/2 + 1
    for i := 0; i < k; i++ {
        num = arr[i]
        if _, ok := kChunkCount[num]; !ok {
            kChunkCount[num] = 0
        }
        kChunkCount[num]++

        // dump(kChunkCount, numCount, 0, k-1,
        computeLeader(kChunkCount, numCount, p
    }

    ans := make(map[int]bool)
    if len(previousKAns) == 1 {
        for ansKey := range previousKAns {
            // fmt.Printf("Leader %d for range
            ans[ansKey] = true
        }
    }

    // Now start evaluating each k-chunk for
    // answer. For each chunk, we discard the
    // chunk and add it back to the global cou
    // entry of the current chunk from the gl
    // k-chunk count.
    for begin := 1; begin <= n-k; begin++ {
        // Discard previous beginning.
        kChunkCount[arr[begin-1]]--
        numCount[arr[begin-1]]++
        // Add new end.
        kChunkCount[arr[begin+k-1]]++
        numCount[arr[begin+k-1]]--

        // Recompute the two ends for the cur
        // previous beginning.
        num = arr[begin-1]
        //dump(kChunkCount, numCount, begin, 
        computeLeader(kChunkCount, numCount, 
        computeLeader(kChunkCount, numCount, 

        // Then the new end.
        num = arr[begin+k-1]
        // dump(kChunkCount, numCount, begin,
        computeLeader(kChunkCount, numCount, 
        computeLeader(kChunkCount, numCount, 

        // // Also check if num - 1 was in th
        // if _, ok := previousKAns[num-1]; o
        //  leaderCount = kChunkCount[num-1]
        //  leaderCount += numCount[num]
        //  dump(kChunkCount, numCount, begin
        //  if leaderCount < halfCount {
        //      delete(previousKAns, num-1)
        //  }
        // }

        if len(previousKAns) == 1 {
            for ansKey := range previousKAns 
                // fmt.Printf("Leader %d for 
                ans[ansKey] = true
            }
        }
    }

    keys := make([]int, 0, len(ans))
    for ansKey := range ans {
        keys = append(keys, ansKey)
    }
    sort.Ints(keys)

    return keys
}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: O(M)

expand all Example tests

▶ example1 ✔ OK
first example test

▶ example2 ✔ OK
second example test

expand all Correctness tests

▶ corner_cases ✔ OK
corner cases

▶ one_leader only one leader is possible	✔ OK
▶ occurs_in_half value occurs in exactly half of elements	✔ OK
▶ two_leaders there are 2 possible leaders	✔ OK
▶ no_leaders no possible leader	✔ OK
expand all Performance tests	
▶ medium_tests1 medium tests (N = 10000, M = 100)	✔ OK
▶ medium_tests2 medium tests(N >= 20000, M=30000)	✔ OK
▶ max_tests1 max possible N, small K	✔ OK
▶ max_tests2 max possible N, big K	✔ OK

The PDF version of this report that may be downloaded on top of this site may contain sensitive data including personal information. For security purposes, we recommend you remove it from your system once reviewed.