Codility_

Candidate Report: trainingH2HFH5-VEG

Check out Codility training tasks

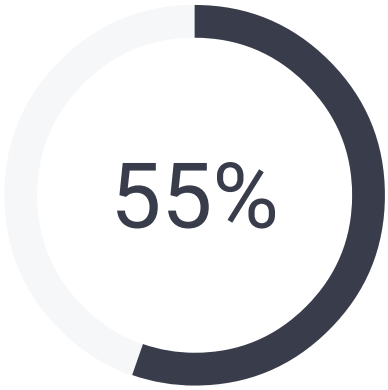
Test Name:

Summary Timeline

Tasks summary

Task	Time spent	Score
MaxCounters C	44 min	55%

Total score



Tasks Details

Medium

1. MaxCounters

Calculate the values of counters after applying all alternating operations: increase counter by 1; set value of all counters to current maximum.

Task Score	Correctness	Performance
55%	75%	40%

Task description

You are given N counters, initially set to 0, and you have two possible operations on them:

- *increase(X)* − counter X is increased by 1,
- *max counter* − all counters are set to the maximum value of any counter.

A non-empty array A of M integers is given. This array represents consecutive operations:

- if A[K] = X, such that 1 ≤ X ≤ N, then operation K is increase(X),
- if A[K] = N + 1 then operation K is max counter.

For example, given integer N = 5 and array A such that:

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

Solution

Programming language used:	C
Total time used:	44 minutes
Effective time used:	44 minutes
Notes:	*not defined yet*

Task timeline

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

the values of the counters after each consecutive operation will be:

```
(0, 0, 1, 0, 0)
(0, 0, 1, 1, 0)
(0, 0, 1, 2, 0)
(2, 2, 2, 2, 2)
(3, 2, 2, 2, 2)
(3, 2, 2, 3, 2)
(3, 2, 2, 4, 2)
```

The goal is to calculate the value of every counter after all operations.

Assume that the following declarations are given:

```
struct Results {
  int * C;
  int L; // Length of the array
};
```

Write a function:

```
struct Results solution(int N, int A[], int M);
```

that, given an integer N and a non-empty array A consisting of M integers, returns a sequence of integers representing the values of the counters.

Result array should be returned as a structure `Results`.

For example, given:

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

the function should return [3, 2, 2, 4, 2], as explained above.

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

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

07:26:37 08:09:58

Code: 08:09:58 UTC, c, final, score: **55** show code in pop-up

```c
// you can write to stdout for debugging purposes,
// printf("this is a debug message\n");
#include <string.h>
#define set_max(c, n, max) for(int j =0; j<n; j++)

struct Results solution(int N, int A[], int M) {
    struct Results result;
    // write your code in C99 (gcc 6.2.0)
    int i = 0, max = 0;

    result.C = malloc(sizeof(int)*N);
    memset(result.C, 0, sizeof(int)*N);

    for (i=0; i<M; i++){
        if (A[i]>=N)
            set_max(result.C, N, max);
        else{
            result.C[A[i]-1]++;
            if (result.C[A[i]-1] > max)
                max = result.C[A[i]-1];
        }
        // printf("C:[");
        // for(int k=0; k<N; k++)
        //     printf(" %d", result.C[k]);
        // printf("] \n");
    }
    result.L = N;
    return result;
}
```

Analysis summary

The following issues have been detected: wrong answers, timeout errors.

For example, for the input `(1, [1])` the solution returned a wrong answer (got [0] expected [1]).

Analysis

collapse all — Example tests

▼ example example test	✓ OK	
1. 0.001 s OK		

collapse all — Correctness tests

▼ extreme_small all max_counter operations	✓ OK	
1. 0.001 s OK		
▼ single only one counter	✗ WRONG ANSWER	got [0] expected [3]
1. 0.001 s WRONG ANSWER, got [0] expected [3]		
2. 0.001 s WRONG ANSWER, got [0] expected [1]		

▼ small_random1 small random test, 6 max_counter operations	✓ OK	
1. 0.001 s OK		
▼ small_random2 small random test, 10 max_counter operations	✓ OK	
1. 0.001 s OK		

collapse all Performance tests

▼ medium_random1 medium random test, 50 max_counter operations	✓ OK	
1. 0.001 s OK		
▼ medium_random2 medium random test, 500 max_counter operations	✗ WRONG ANSWER got [326, 326, 326, 326, 32.. expected [325, 325, 325, 325, 32..	
1. 0.001 s WRONG ANSWER, got [326, 326, 326, 326, 32.. expected [325, 325, 325, 325, 32..		
▼ large_random1 large random test, 2120 max_counter operations	✓ OK	
1. 0.028 s OK		
▼ large_random2 large random test, 10000 max_counter operations	✗ TIMEOUT ERROR running time: 0.112 sec., time limit: 0.100 sec.	
1. 0.112 s TIMEOUT ERROR, running time: 0.112 sec., time limit: 0.100 sec.		
▼ extreme_large all max_counter operations	✗ TIMEOUT ERROR running time: 3.700 sec., time limit: 0.100 sec.	
1. 3.700 s TIMEOUT ERROR, running time: 3.700 sec., time limit: 0.100 sec.		
2. 0.008 s OK		