closed

Demo ticket

Session
ID: demoPCSAZK-8CN
Time limit: 120 min.

Status: closed
Started on: 2014-01-05 05:44 UTC

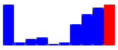
Score:

100

of 100

★★ 1. MaxCounters

score: 100 of 100



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

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 zero-indexed 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
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.

Write a function:

```
class Solution { public int[] solution(int N, int[] A); }
```

that, given an integer N and a non-empty zero-indexed array A consisting of M integers, returns a sequence of integers representing the values of the counters.
The sequence should be returned as:

- a structure Results (in C), or
- a vector of integers (in C++), or
- a record Results (in Pascal), or
- an array of integers (in any other programming language).

For example, given:

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

Solution

Programming language used: C#

Total time used: 57 minutes (?)

Effective time used: 1 minutes (?)

Notes: correct functionality and scalability

Task timeline What is it?

05:44:51 06:41:28

Code: 06:41:27 UTC, cs, final, score: 100.00

```
using System;
// you can also use other imports, for example:
// using System.Collections.Generic;
class Solution {
        public int[] solution(int N, int[] A)
        {
            var length = A.Length;
            if (N < 1 || length < 1 || N > 100000 || length > 100000) throw new ArgumentOutOfRangeException();
            // write your code in C# with .NET 2.0
            var result = new int[N];
            var maxValue = 0;
            var resetValue = 0;
            var hasReset = false;
            for (var count = 0; count < length; count++)
            {
                if (A[count] <= N)
                {
                    // Set to maximum value if a reset has been encountered
                    if (hasReset && (resetValue > result[A[count] - 1]))
                     result[A[count] - 1] = resetValue;

                    result[A[count] - 1] += 1;
```

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

- 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].

Complexity:

- expected worst-case time complexity is O(N+M);
- 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.

```

          // Get new maximum value
          if (result[A[count] - 1] > maxValue)
            maxValue = result[A[count] - 1];
        }
        else
        {
          resetValue = maxValue;
          hasReset = true;
        }
      }

      for (var count = 0; count < N; count++)
      {
        if (hasReset && resetValue > result[count])
          result[count] = resetValue;
      }

      return result;
    }
}
```

Analysis

Detected time complexity:

O(N + M)

test	time	result
example example test	0.080 s.	**OK**
extreme_small all max_counter operations	0.070 s.	**OK**
single only one counter	0.080 s.	**OK**
small_random1 small random test, 6 max_counter operations	0.080 s.	**OK**
small_random2 small random test, 10 max_counter operations	0.080 s.	**OK**
medium_random1 medium random test, 50 max_counter operations	0.080 s.	**OK**
medium_random2 medium random test, 500 max_counter operations	0.080 s.	**OK**
large_random1 large random test, 2120 max_counter operations	0.090 s.	**OK**
large_random2 large random test, 10000 max_counter operations	0.110 s.	**OK**
extreme_large all max_counter operations	0.110 s.	**OK**

Get acco