Codility_

CodeCheck Report: trainingMK9GWS-DY9

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Fish C# ⚠	4 min	100%

Total score

100%

Tasks Details

Easy

1. Fish

N voracious fish are moving along a river. Calculate how many fish are alive.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are given two non-empty arrays A and B consisting of N integers. Arrays A and B represent N voracious fish in a river, ordered downstream along the flow of the river.

The fish are numbered from 0 to N − 1. If P and Q are two fish and P < Q, then fish P is initially upstream of fish Q. Initially, each fish has a unique position.

Fish number P is represented by A[P] and B[P]. Array A contains the sizes of the fish. All its elements are unique. Array B contains the directions of the fish. It contains only 0s and/or 1s, where:

- 0 represents a fish flowing upstream,
- 1 represents a fish flowing downstream.

If two fish move in opposite directions and there are no other (living) fish between them, they will eventually meet each other. Then only one fish can stay alive − the larger fish eats the smaller one. More precisely, we say that two fish P and Q meet each other

Solution

Programming language used:	C#
Total time used:	4 minutes
Effective time used:	4 minutes
Notes:	*not defined yet*

Task timeline

⏮ ⏪ ▶ ■ ⏩ ⏭

07:32:41 07:36:11

when P < Q, B[P] = 1 and B[Q] = 0, and there are no living fish between them. After they meet:

- If A[P] > A[Q] then P eats Q, and P will still be flowing downstream,
- If A[Q] > A[P] then Q eats P, and Q will still be flowing upstream.

We assume that all the fish are flowing at the same speed. That is, fish moving in the same direction never meet. The goal is to calculate the number of fish that will stay alive.

For example, consider arrays A and B such that:

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

Initially all the fish are alive and all except fish number 1 are moving upstream. Fish number 1 meets fish number 2 and eats it, then it meets fish number 3 and eats it too. Finally, it meets fish number 4 and is eaten by it. The remaining two fish, number 0 and 4, never meet and therefore stay alive.

Write a function:

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

that, given two non-empty arrays A and B consisting of N integers, returns the number of fish that will stay alive.

For example, given the arrays shown above, the function should return 2, as explained above.

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

- 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];
- each element of array B is an integer that can have one of the following values: 0, 1;
- the elements of A are all distinct.

Code: 07:36:11 UTC, cs, final, score: 100

show code in pop-up

```
using System;
using System.Collections.Generic;

/**
 * 7.2 - Fish
 * Paulo Santos
 * 09.Dec.2022
 */
class Solution {

    private class Fish {

        public Fish(int s, int d) {
            this.Size = s;
            this.Direction = d;
        }

        public int Size      {get; private set;}
        public int Direction {get; private set;}
    }

    public int solution(int[] A, int[] B) {

        var sEats = new Stack<Fish>();

        sEats.Push(new Fish(A[0], B[0]));
        for (var i = 1; i < A.Length; i++) {
            if (sEats.Peek().Direction == B[i]) {
                /*
                 * Add to the stack
                 */
                sEats.Push(new Fish(A[i], B[i]));
                continue;
            }
            /*
            * If the top fist is flowing upstream
            * it's not the right condition to see
            * who's eating who.
            */
            if (sEats.Peek().Direction == 0) {
                sEats.Push(new Fish(A[i], B[i]));
            }
            else {
                /*
                * Figure out who's eating who
                */
                while (sEats.Count > 0) {
                    /*
                    * Top fish swimming in the same
                    * They can't eat each other
                    */
                    if (sEats.Peek().Direction == B
                        sEats.Push(new Fish(A[i], B
                        break;
                    }
                    else {
                        /*
                        * Top fish eats current fis
                        */
                        if (sEats.Peek().Size > A[i
                            /*
                            * yes.
                            */
                            break;
                        }
                        else {
                            /*
                            * nope.
                            */
                            sEats.Pop();
                            continue;
                        }
                    }
                }
                if (sEats.Count == 0) {
                    sEats.Push(new Fish(A[i], B[i])
                }
            }
        }

        return sEats.Count;
    }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example example test	✓ OK
expand all	**Correctness tests**	
▶	extreme_small 1 or 2 fishes	✓ OK
▶	simple1 simple test	✓ OK
▶	simple2 simple test	✓ OK
▶	small_random small random test, N = ~100	✓ OK
expand all	**Performance tests**	
▶	medium_random small medium test, N = ~5,000	✓ OK
▶	large_random large random test, N = ~100,000	✓ OK
▶	extreme_range1 all except one fish flowing in the same direction	✓ OK
▶	extreme_range2 all fish flowing in the same direction	✓ OK