Codility_

CodeCheck Report: trainingJCVNSZ-S3C

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Fish Java 8	2 min	100%

Total score

Tasks Details

1. Fish

Easy

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:

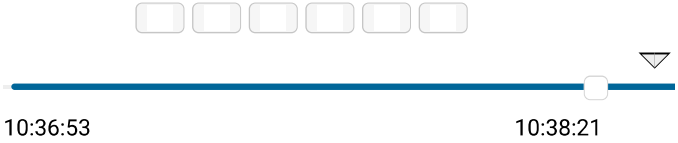
- 0 represents a fish flowing upstream,
- 1 represents a fish flowing downstream.

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:	Java 8
Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Task timeline



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: 10:38:21 UTC, java, final, score: **100** show code in pop-up

```
1  // you can also use imports, for example:
2  // import java.util.*;
3  import java.util.ArrayList;
4  // you can write to stdout for debugging purpos
5  // System.out.println("this is a debug message'
6
7  class Solution {
8      public static void push(ArrayList<Integer>
9                  intList.add(i);
10         }
11
12         public static int pop(ArrayList<Intege
13                 return (intList.isEmpty()?-1:i
14         }
15
16     public int solution(int[] A, int[] B) {
17         // write your code in Java SE 8
18         ArrayList<Integer> upstreamFish = new
19                 int aliveFishCount=0;
20
21                 for (int i = 0; i < B.length;
22                         if(B[i]==0) {
23                                 if(upstreamFis
24                                         aliveF
25                                 }else {
26                                         boolea
27                                         do {
28
29
30
31
32
33
34
35
36
37
38                                         } whil
39                                 }
40
41                         }else {
42                                 push(upstreamF
43                         }
44                 }
45
46                 aliveFishCount = aliveFishCoun
47                 return aliveFishCount;
48     }
49 }
```

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**

▶	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