Codility_

CodeCheck Report: training5V3Q38-32K

Test Name:

Check out Codility training tasks

Summary   Timeline

Tasks summary

Task	Time spent	Score
Ladder ⚠ C#	2 min	100%

Total score

100%

Tasks Details

Medium

1. Ladder

Count the number of different ways of climbing to the top of a ladder.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You have to climb up a ladder. The ladder has exactly N rungs, numbered from 1 to N. With each step, you can ascend by one or two rungs. More precisely:

- with your first step you can stand on rung 1 or 2,
- if you are on rung K, you can move to rungs K + 1 or K + 2,
- finally you have to stand on rung N.

Your task is to count the number of different ways of climbing to the top of the ladder.

For example, given N = 4, you have five different ways of climbing, ascending by:

- 1, 1, 1 and 1 rung,
- 1, 1 and 2 rungs,
- 1, 2 and 1 rung,
- 2, 1 and 1 rungs, and
- 2 and 2 rungs.

Solution

Programming language used:	C#
Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Task timeline

19:48:35   19:50:09

Given N = 5, you have eight different ways of climbing, ascending by:

- 1, 1, 1, 1 and 1 rung,
- 1, 1, 1 and 2 rungs,
- 1, 1, 2 and 1 rung,
- 1, 2, 1 and 1 rung,
- 1, 2 and 2 rungs,
- 2, 1, 1 and 1 rungs,
- 2, 1 and 2 rungs, and
- 2, 2 and 1 rung.

The number of different ways can be very large, so it is sufficient to return the result modulo 2^P, for a given integer P.

Write a function:

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

that, given two non-empty arrays A and B of L integers, returns an array consisting of L integers specifying the consecutive answers; position I should contain the number of different ways of climbing the ladder with A[I] rungs modulo $2^{B[I]}$.

For example, given L = 5 and:

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

the function should return the sequence [5, 1, 8, 0, 1], as explained above.

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

- L is an integer within the range [1..50,000];
- each element of array A is an integer within the range [1..L];
- each element of array B is an integer within the range [1..30].

Code: 19:50:09 UTC, cs, final, score: **100**

show code in pop-up

```
using System;

/**
 * 13.2 - Ladder
 * Paulo Santos
 * 07.Dec.2023
 */
class Solution {
    public int[] solution(int[] A, int[] B) {
        var f = new int[A.Length + 1];
        f[0] = 1;
        f[1] = 1;
        var MAX = 1<<30;

        for (var i = 2; i < f.Length; ++i) {
            f[i] = f[i-1] + f[i-2];
            f[i] = f[i] % MAX;
        }

        var res = new int[A.Length];

        for (var i = 0; i < A.Length; ++i) {
            res[i] = f[A[i]] % (1 << B[i]);
        }

        return res;
    }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all **Example tests**

▶ example ✓ OK
example test

expand all **Correctness tests**

▶ extreme ✓ OK
extreme small values

▶ small_functional ✓ OK
small functional

▶ small ✓ OK
small tests

▶ small_random ✓ OK
small random, length = ~100

expand all **Performance tests**

▶ medium_random ✓ OK
medium random, length = ~1,000

▶ large_range ✓ OK
large range, length = ~30,000

▶	large_random large random, length = ~30,000	✓ OK
▶	extreme_large all max size of the ladder	✓ OK