Codility_

CodeCheck Report: trainingGMDTNF-TTN

Test Name:

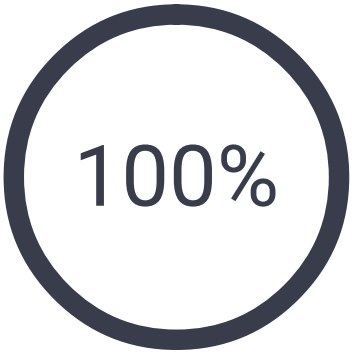
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Ladder Java 8 ⚠	3 min	100%

Total score



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.

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

Solution

Programming language used:	Java 8
Total time used:	3 minutes
Effective time used:	3 minutes
Notes:	*not defined yet*

Task timeline

21:21:34 21:24:04

Code: 21:24:03 UTC, java, final, score: **100** show code in pop-up

- 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].


```
class Solution {
    public int[] solution(int[] A, int[] B) {
        int N = A.length;
        int maxA = 0;
        int maxB = 0;

        for (int i = 0; i < N; i++) {
            maxA = Math.max(maxA, A[i]);
            maxB = Math.max(maxB, B[i]);
        }

        int[] modValues = new int[maxB + 1];
        modValues[0] = 1;

        for (int b = 1; b <= maxB; b++) {
            modValues[b] = modValues[b - 1] *
        }


        long[][] fibonacciNumbers = new long[
        int[] maxRungNumbers = new int[maxB +


        int[] result = new int[A.length];
        for (int k = 0; k < A.length; k++) {
            int b = B[k];
            int a = A[k];
            if (maxRungNumbers[b] < a) {
                fibonacciNumbers[b][0] = 1L;
                fibonacciNumbers[b][1] = 1L;

                for (int i = Math.max(2, maxR
                    fibonacciNumbers[b][i] =
                }

                maxRungNumbers[b] = a;
            }
            result[k] = (int)fibonacciNumbers
        }

        return result;
    }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

collapse all	Example tests	
▼ example example test		✔ OK
1. 0.004 s OK		
collapse all	Correctness tests	
▼ extreme extreme small values		✔ OK
1. 0.004 s OK		
▼ small_functional small functional		✔ OK

1. 0.008 s **OK**

▼ small ✔ OK

small tests

1. 0.008 s **OK**

▼ small_random ✔ OK

small random, length = ~100

1. 0.008 s **OK**

collapse all Performance tests

▼ medium_random ✔ OK

medium random, length = ~1,000

1. 0.020 s **OK**

▼ large_range ✔ OK

large range, length = ~30,000

1. 0.492 s **OK**

▼ large_random ✔ OK

large random, length = ~30,000

1. 0.464 s **OK**

▼ extreme_large ✔ OK

all max size of the ladder

1. 0.812 s **OK**