Codility_

CodeCheck Report: trainingQQR3R8-F4M

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
FibFrog C# ⚠	25 min	100%

Total score

100%

Tasks Details

Medium

1. FibFrog

Count the minimum number of jumps required for a frog to get to the other side of a river.

Task Score	Correctness	Performance
100%	100%	100%

Task description

The Fibonacci sequence is defined using the following recursive formula:

```
    F(0) = 0
    F(1) = 1
    F(M) = F(M - 1) + F(M - 2) if M >= 2
```

A small frog wants to get to the other side of a river. The frog is initially located at one bank of the river (position −1) and wants to get to the other bank (position N). The frog can jump over any distance F(K), where F(K) is the K-th Fibonacci number. Luckily, there are many leaves on the river, and the frog can jump between the leaves, but only in the direction of the bank at position N.

The leaves on the river are represented in an array A consisting of N integers. Consecutive elements of array A represent consecutive positions from 0 to N − 1 on the river. Array A contains only 0s and/or 1s:

- 0 represents a position without a leaf;

Solution

Programming language used:	C#
Total time used:	25 minutes ?
Effective time used:	25 minutes ?
Notes:	*not defined yet*

Task timeline ?

19:00:47 19:25:26

- 1 represents a position containing a leaf.

The goal is to count the minimum number of jumps in which the frog can get to the other side of the river (from position −1 to position N). The frog can jump between positions −1 and N (the banks of the river) and every position containing a leaf.

For example, consider array A such that:

```
A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 1
A[4] = 1
A[5] = 0
A[6] = 1
A[7] = 0
A[8] = 0
A[9] = 0
A[10] = 0
```

The frog can make three jumps of length F(5) = 5, F(3) = 2 and F(5) = 5.

Write a function:

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

that, given an array A consisting of N integers, returns the minimum number of jumps by which the frog can get to the other side of the river. If the frog cannot reach the other side of the river, the function should return −1.

For example, given:

```
A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 1
A[4] = 1
A[5] = 0
A[6] = 1
A[7] = 0
A[8] = 0
A[9] = 0
A[10] = 0
```

the function should return 3, as explained above.

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

- N is an integer within the range [0..100,000];
- each element of array A is an integer that can have one of the following values: 0, 1.

Code: 19:25:26 UTC, cs, final, score: 100

show code in pop-up

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

/**
 * 13.1 - Fib Frog
 * Paulo Santos
 * 06.Jan.2023
 */
class Solution {
    public int solution(int[] A) {

        /*
         * Convert the array in list to facilitate
         */
        var lstA = A.ToList();
        lstA.Add(1); // The opposite bank

        /*
         * Prepares an array for the shortest path
         */
        var lstPath = new List<int>();
        for(var i = 0; i < lstA.Count; i++)
            lstPath.Add(-1);

        /*
         * Calculates the Fibonacci series of A.Len
         */
        var lstFib = new List<int>();
        lstFib.Add(1);
        lstFib.Add(1);
        var fib = 2;
        var cur = 1;
        while(fib <= lstA.Count) {
            lstFib.Add(fib);
            cur += 1;
            fib = lstFib[cur] + lstFib[cur - 1];
        }

        for (var i = 0; i < lstFib.Count; i++)
            if (lstA[lstFib[i] - 1] == 1)
                lstPath[lstFib[i] - 1] = 1;

        /*
         * Calculates the shortest path
         */
        for(var i = 0; i < lstA.Count; i++) {
            if ((lstA[i] == 1) &&
                (lstPath[i] == -1)) {
                var minPrevPos = -1;
                var minDist = int.MaxValue;
                for (var j = 0; j < lstFib.Count; j
                    var prevPos = i - lstFib[j];
                    if (prevPos < 0)
                        break;
                    if ((lstPath[prevPos] > 0) &&
                        (minDist > lstPath[prevPos]
                        minPrevPos = prevPos;
                        minDist = lstPath[prevPos];
                    }
                }
                if (minPrevPos > -1)
                    lstPath[i] = minDist + 1;
            }
        }

        return lstPath[lstPath.Count - 1];
    }

}
```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all	Example tests	
▶	example example test	✓ OK
expand all	**Correctness tests**	
▶	extreme_small_ones empty array and all ones	✓ OK
▶	extreme_small_zeros all zeros	✓ OK
▶	simple_functional simple functional tests	✓ OK
▶	small_random small random test, length = ~100	✓ OK
▶	small_cyclic small cyclic test, length = ~500	✓ OK
▶	small_fibonacci small Fibonacci word test, length = 610	✓ OK
expand all	**Performance tests**	
▶	medium_random medium random test, length = ~5,000	✓ OK
▶	medium_thue_morse medium Thue-Morse sequence, lenght = 2^13	✓ OK
▶	large_big_result large test with big result, length = ~100,000	✓ OK
▶	large_cyclic large cyclic test, length = ~100,000	✓ OK
▶	large_random large random test, length = ~100,000	✓ OK
▶	extreme_large_ones_zeros all zeros / ones, length = ~100,000	✓ OK