Codility_

CodeCheck Report: training3JRU2B-ED7

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task		Time spent	Score
FibFrog Python	⚠	8 min	100%

Total score

100%

Tasks Details

		Task Score	Correctness	Performance
Medium	1. FibFrog Count the minimum number of jumps required for a frog to get to the other side of a river.	100%	100%	100%

Task description

The Fibonacci sequence is defined using the following recursive formula:

```
F(0) = 0
F(1) = 1
```

Solution

Programming language used:	Python
Total time used:	8 minutes ?

```
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;
- 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:

```
def solution(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.

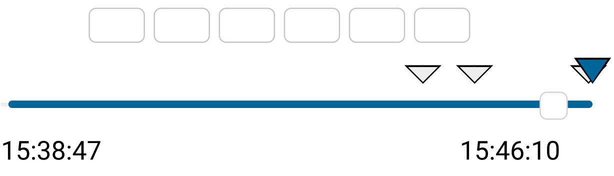
For example, given:

```
A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 1
```

Effective time used: 8 minutes

Notes: *not defined yet*

Task timeline



Code: 15:46:09 UTC, py, final, score: **100**

show code in pop-up

```
# you can write to stdout for debuggi
# print("this is a debug message")

def solution(A):
    # Implement your solution here
    # pass
    N = len(A)

    # Add an extra leaf on the other
    A.append(1)

    # Generate the Fibonacci sequence
    fibonacci = [0, 1]
    while fibonacci[-1] <= N + 1:
        fibonacci.append(fibonacci[-1

    # Remove the unnecessary first tw
    fibonacci = fibonacci[2:]

    # Initialize the dynamic programm
    dp = [-1] * (N + 1)

    # Calculate the minimum number of
    for i in range(N + 1):
        if A[i] == 0:
            continue

        for fib in fibonacci:
            if fib - 1 == i:
                dp[i] = 1
            elif fib - 1 < i and dp[i
                if dp[i] == -1 or dp[
                    dp[i] = dp[i - fi

    return dp[N]

```

Analysis summary

The solution obtained perfect score.

Analysis

```
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.

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