Codility_

## CodeCheck Report: trainingQF3KYN-HAF

Test Name:

Check out Codility training tasks

Summary Timeline

### Tasks summary

| Task | Time spent | Score |
|---|---|---|
| FrogRiverOne ⚠<br>Python | 37 min | 100% |

### Total score

100%

## Tasks Details

1. FrogRiverOne

Easy

Find the earliest time when a frog can jump to the other side of a river.

| Task Score | Correctness | Performance |
|---|---|---|
| 100% | 100% | 100% |

### Task description

A small frog wants to get to the other side of a river. The frog is initially located on one bank of the river (position 0) and wants to get to the opposite bank (position X+1). Leaves fall from a tree onto the surface of the river.

You are given an array A consisting of N integers representing the falling leaves. A[K] represents the position where one leaf falls at time K, measured in seconds.

The goal is to find the earliest time when the frog can jump to the other side of the river. The frog can cross only when leaves appear at every position across the river from 1 to X (that is, we want to find the earliest moment when all the positions from 1 to X are covered by leaves). You

### Solution

| | |
|---|---|
| Programming language used: | Python |
| Total time used: | 37 minutes ? |
| Effective time used: | 37 minutes ? |
| Notes: | *not defined yet* |

### Task timeline ?

may assume that the speed of the current in the river is negligibly small, i.e. the leaves do not change their positions once they fall in the river.

For example, you are given integer X = 5 and array A such that:

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

In second 6, a leaf falls into position 5. This is the earliest time when leaves appear in every position across the river.

Write a function:

```
def solution(X, A)
```

that, given a non-empty array A consisting of N integers and integer X, returns the earliest time when the frog can jump to the other side of the river.

If the frog is never able to jump to the other side of the river, the function should return −1.

For example, given X = 5 and array A such that:

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

the function should return 6, as explained above.

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

- N and X are integers within the range [1..100,000];
- each element of array A is an integer within the range [1..X].

11:05:55 11:42:14

Code: 11:42:14 UTC, py, final, score: **100** show code in pop-up

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

def solution(X, A):
    # Implement your solution here
    # pass
    not_covered = [True] * X
    covered_count = 0

    for seconds, leaf in enumerate(A):
        if leaf <= X and not_covered[lea
            not_covered[leaf - 1] = Fals
            covered_count += 1
            if covered_count == X:
                return seconds

    return -1
```

## 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**

▶ simple ✔ OK
simple test

▶ single ✔ OK
single element

▶ extreme_frog ✔ OK
frog never across the river

▶ small_random1 ✔ OK
3 random permutation, X = 50

▼ small_random2 ✔ OK
5 random permutation, X = 60

1. 0.012 s **OK**

▼ extreme_leaves ✔ OK
all leaves in the same place

1. 0.012 **OK**

s

2. 0.012 s **OK**

collapse all

### Performance tests

▼ medium_random ✔ **OK**
6 and 2 random permutations, X = ~5,000

1. 0.020 s **OK**
2. 0.016 s **OK**

▼ medium_range ✔ **OK**
arithmetic sequences, X = 5,000

1. 0.016 s **OK**

▼ large_random ✔ **OK**
10 and 100 random permutation, X = ~10,000

1. 0.052 s **OK**
2. 0.044 s **OK**

▼ large_permutation ✔ **OK**
permutation tests

1. 0.060 s **OK**
2. 0.064 s **OK**

▼ large_range ✔ **OK**
arithmetic sequences, X = 30,000

1. 0.032 s **OK**