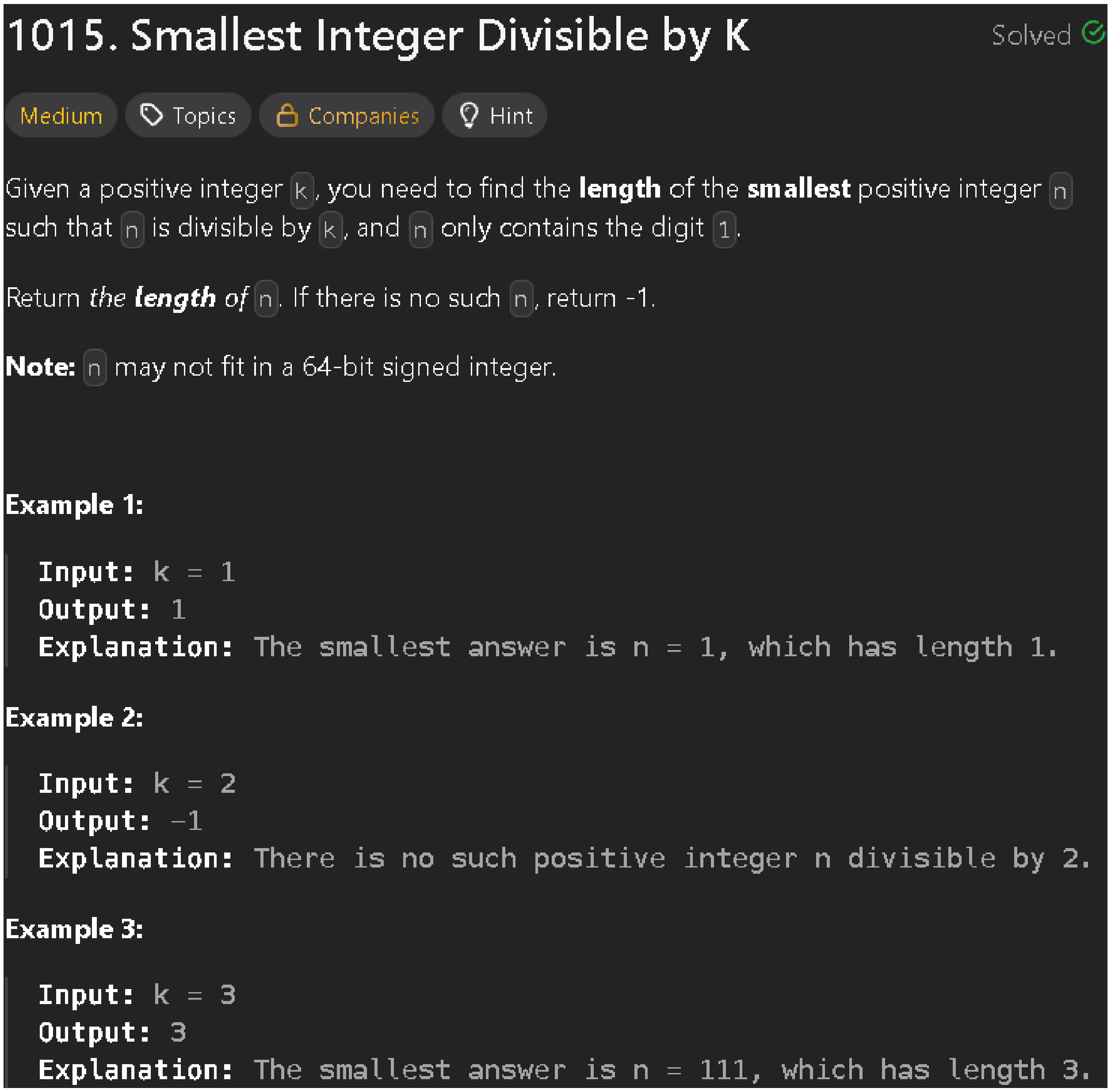


1015. Smallest Integer Divisible by K

Solved

Medium · Topics · Companies · Hint

Given a positive integer `k`, you need to find the **length** of the **smallest** positive integer `n` such that `n` is divisible by `k`, and `n` only contains the digit `1`.

Return *the* ***length*** *of* `n`. If there is no such `n`, return -1.

Note: `n` may not fit in a 64-bit signed integer.

Example 1:

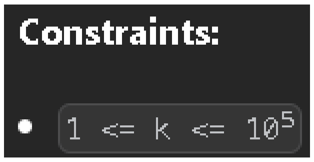
```
Input: k = 1
Output: 1
Explanation: The smallest answer is n = 1, which has length 1.
```

Example 2:

```
Input: k = 2
Output: -1
Explanation: There is no such positive integer n divisible by 2.
```

Example 3:

```
Input: k = 3
Output: 3
Explanation: The smallest answer is n = 111, which has length 3.
```



Constraints:

- `1 <= k <= 10^5`

Python:

```
class Solution(object):
    def smallestRepunitDivByK(self, k):
        # if k % 2 == 0 or k % 5 == 0: return -1  # this trick may save a little time
        hit, n, ans = [False] * k, 0, 0
        while True:  # at most k times, because 0 <= remainder < k
```

```
            ans, n = ans + 1, (n * 10 + 1) % k  # we only focus on whether to divide, so we only need
to keep the remainder.
            if n == 0: return ans  # can be divisible
            if hit[n]: return -1  # the remainder of the division repeats, so it starts to loop that means it
cannot be divisible.
            hit[n] = True
```

JavaScript:

```
var smallestRepunitDivByK = function(K) {
    let count = 1
    let start = 1
    let set = new Set()
    while (start) {
        let r = start % K
        if ( r === 0 ) {
            break
        }
        if ( set.has(r) === true ) {
            return -1
        } else {
            set.add(r)
            start = r * 10 + 1
            count++
        }

    }
    return count
};
```

Java:

```
class Solution {
    public int smallestRepunitDivByK(int k) {
        // if (k % 2 == 0 || k % 5 == 0) return -1;  // this trick may save a little time
        boolean[] hit = new boolean[k];
        int n = 0, ans = 0;
        while (true) { // at most k times, because 0 <= remainder < k
            ++ ans;
            n = (n * 10 + 1) % k; // we only focus on whether to divide, so we only need to keep the
remainder.
            if (n == 0) return ans; // can be divisible
            if (hit[n]) return -1; // the remainder of the division repeats, so it starts to loop that means
it cannot be divisible.
            hit[n] = true;
        }
    }}
```