

Problem — Grasshopper on a Line (CF 1837A)

A grasshopper starts at position **0** on a number line.

You are given:

- A target position **x**
- An integer **k**

The grasshopper can make jumps of **any integer length**, with one restriction:

It **cannot** make a jump whose length is divisible by **k**

Your task is to determine:

- The **minimum number of jumps**
- The **length of each jump**

so that the grasshopper reaches exactly position **x**.

Key Observations

1. Direct Jump is Allowed

If the jump of length **x** is **NOT divisible by k**, then:

- The grasshopper can jump directly from $0 \rightarrow x$
- Only **1 jump** is needed

Condition:

$$x \bmod k \neq 0$$

2. Direct Jump is NOT Allowed

If:

$$x \bmod k = 0$$

Then jumping directly is forbidden.

So we split the movement into **two jumps**, such that:

- Each jump length is **not divisible by k**
- Their sum = **x**

A simple safe split:

$$x = (x - (k+1)) + (k+1)$$

Both parts:

- $(k+1)$ is **never divisible by k**
- $(x - (k+1))$ is also **not divisible by k**

So the grasshopper reaches the target in **2 jumps**.

3. Special Case — When $(x < k)$

Since any number smaller than (k) cannot be divisible by (k) :

- Direct jump is always valid
 - Answer = **1 jump**
-

Final Strategy

Condition	Jumps Needed	Explanation
$(x < k)$	1	Direct jump is valid
$(x \bmod k \neq 0)$	1	Direct jump allowed
$(x \bmod k = 0)$	2	Split into safe jumps

Why 2 Jumps Always Work?

If (x) is divisible by (k) :

We avoid multiples of (k) by breaking:

$$x = (x - k - 1) + (k + 1)$$

- $(k+1) \rightarrow$ not divisible by (k)
- $(x - k - 1) \rightarrow$ also not divisible by (k)

So both jumps are valid.

Clean Pseudocode

```
int t = fs.readInt();
```

```
while (t-- > 0) {
```

```
    int x = fs.readInt();
```

```
    int k = fs.readInt();
```

```
    if (x % k != 0) {
```

```
        // Direct jump possible
```

```
        System.out.println(1);
```

```
        System.out.println(x);
```

```
    }
```

```
    else {
```

```
        // Need 2 jumps
```

```
        System.out.println(2);
```

```
        System.out.println(x - (k + 1) + " " + (k + 1));
```

```
    }
```

```
}
```

Complexity

- Time: **$O(1)$** per test case
 - Space: **$O(1)$**
-

Intuition Summary

- If the destination itself is legal \rightarrow jump once.
 - If not \rightarrow split into two non-multiples of k .
 - Two jumps are always sufficient (except trivial direct case).
-

This frames your solution more logically:

Restriction check \rightarrow Direct move \rightarrow Otherwise split safely.