## Problem

Last year, the Infinite House of Pancakes introduced a new kind of pancake. It has a happy face made of chocolate chips on one side (the "happy side"), and nothing on the other side (the "blank side").

You are the head cook on duty. The pancakes are cooked in a single row over a hot surface. As part of its infinite efforts to maximize efficiency, the House has recently given you an oversized pancake flipper that flips exactly **K** consecutive pancakes. That is, in that range of **K** pancakes, it changes every happy-side pancake to a blank-side pancake, and vice versa; it does *not* change the left-to-right order of those pancakes.

You cannot flip fewer than **K** pancakes at a time with the flipper, even at the ends of the row (since there are raised borders on both sides of the cooking surface). For example, you can flip the first **K** pancakes, but not the first **K** - 1 pancakes.

Your apprentice cook, who is still learning the job, just used the old-fashioned single-pancake flipper to flip some individual pancakes and then ran to the restroom with it, right before the time when customers come to visit the kitchen. You only have the oversized pancake flipper left, and you need to use it quickly to leave all the cooking pancakes happy side up, so that the customers leave feeling happy with their visit.

Given the current state of the pancakes, calculate the minimum number of uses of the oversized pancake flipper needed to leave all pancakes happy side up, or state that there is no way to do it.

## Conditions

1 ≤ **T(number of test cases)** ≤ 100.
Every character in **S** is either + or -.
2 ≤ **K** ≤ length of **S**.

## Test cases

To understand a little more about the problem, let's look at some cases that the problem comes along with.

**Some instructions:**

The first line of the input gives the number of test cases, T. T test cases follow. Each consists of one line with a string S and an integer K. S represents the row of pancakes: each of its characters is either + (which represents a pancake that is initially happy side up) or - (which represents a pancake that is initially blank side up).

```
Input               Output

3                   Case #1: 3
---+-++- 3          Case #2: 0
+++++ 4             Case #3: IMPOSSIBLE
-+-+- 4
```

**Case 1**

We want this sequence: ++++++++

The number of pancakes we can flip at one go = **K** = 3

The sequence of pancakes present = **S** = ---+-++-

Let's use a simple positioning system, as described by the following table:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | - | - | + | - | + | + | - |

Since the leftmost pancake is the wrong side up we would like to start there. One interesting aspect to notice is that for the pancakes at the ends there is only one way to do so.

Provided this information, the pancake at 0 has to be flipped and the only way to do so is along with pancakes at 1 and 2. After the flip the arrangement is as shown in the table below.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| + | + | + | + | - | + | + | - |

This flip is convenient as even the pancakes 1 and 2 happen to be flipped to the correct side.

One approach to continue is as follows: Since the pancakes at places 0, 1, 2, and 3 are all happy side up, we could remove that from are area of interest. Now, the area of interest is as shown in the table below.

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| - | + | + | - |

Again, since 4 is the leftmost pancake, there is only one way to flip it----- with pancake at 5 and 6. The new arrangement is shown in the table below.

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| + | - | - | - |

Following the strategy we have employed, since the pancake at four is happy side up we can move it away from our area of focus. Now we have:

| 5 | 6 | 7 |
|---|---|---|
| - | - | - |

This point is crucial for this problem as the number of pancakes remained to flip is equal to the size of the flipper, **K.** In this situation, all the pancakes have to either be all happy side up or blank side up.

In this case, pancakes at 5, 6, and 7 are all blank side up. This results in:

| 5 | 6 | 7 |
|---|---|---|
| + | + | + |

We have now flipped all of them the right side up.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| + | + | + | + | + | + | + | + |

The number of flips equals to 3.

**Case 2**

Since the sequence of pancakes we want (+++++)  and the sequence of the same are the same the number of flips required is 0.

**Case 3**

We want this sequence: +++++

The number of pancakes we can flip at one go = **K** = 4

The sequence of pancakes present = **S** = -+-+-

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| - | + | - | + | - |

There is no way to make the second and third pancakes from the left have the same side up, because any flip flips them both.

Let's see if our strategy works here:

Pancake at 0 is blank side up, so we flip.

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| + | - | + | - | - |

Ignore pancake at 0. Since the number of pancakes remaining is equal to the flipper size, the pancakes all have to be either facing the wrong side or the correct side to make the task possible. Since that is not the case, this task is impossible.

## Simple Observation

If **K** is equal to length of **S**, the sequence has to be uniform, i.e. all pancakes have be either facing the right way or the wrong way.

## Summary of Strategy

The initial status of $p_1$ completely determines what do we need to do with $f_1$ if we want to have any chance of leaving all pancakes happy side up: use $f_1$ if and only if $p_1$ is initially blank side up. After deciding on $f_1$, we can notice that there is only one remaining flip $f_2$ that affects the second to the left pancake $p_2$. Ignoring the pancake with status $p_1$ , the current state of $p_2$ completely determines whether to use $f_2$.

Once **K** equals to the length of **S**, we can easily evaluate if the task can be completed or not.

## Code

```
def flipPancakes(k,n):
    tot = 0
    length = len(k)-(n-1)
    index = 0


    for i in range(0, length):
        if k[i]== '-':

            index = i
            tot += 1
            for j in range(index, index + n):
                if k[j] == '+':
                    k = k[:j] + '-' + k[j+1:]

                else:
                    k = k[:j] + '+' + k[j+1:]

    if k.find('-') == -1:
        return tot
    else:
        return 'IMPOSSIBLE'
```