I assume you know about the XOR operation. In case you don't know, XOR operation of two binary bits is 0 if they are equal, otherwise 1. See the table on the right.

0 XOR 0	0
0 XOR 1	1
1 XOR 0	1
1 XOR 1	0

Initially you will be given a binary string (a string consisting of only `0` or `1`). This will be the first generation. You can generate the $(i+1)$-th generation from the i-th generation by inserting XOR of adjacent bits. So if the i-th generation is `01001` then, the (i+1)-th generation would be `0(1)1(1)0(0)0(1)1` or `011100011` (the bits in the parenthesis are **XOR** of their adjacent bits).

One more example for more than one generations:

First generation	1 0 0 1 0
Second generation	1 1 0 0 0 1 1 1 0
Third generation	1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0
Fourth generation	11011011000000000110110111011010

The spaces are put in the above table for the sake of clarity and to show you how the input is formatted.

You will be given a binary string S, a positive integer G and two positions at the G'th generation of S (original binary string S is the first generation), you need to find the number of `0`'s and `1`'s between the given two positions (inclusively).

Since the G-th generation would be very long, the position will be defined by: p T, where p is an integer ($0 \leq p <$ the length of S) and T is a string of length $(G-1)$ consisting of only '`D`' (down), or/and '`R`' (right). To get the position at the G-th generation, you will start at the p-th position of the first generation and move according to the command in T.

Suppose you are at the i-th generation and the j-th character (0-indexed) in that generation (so initially, $i = 1$, $j = p$).

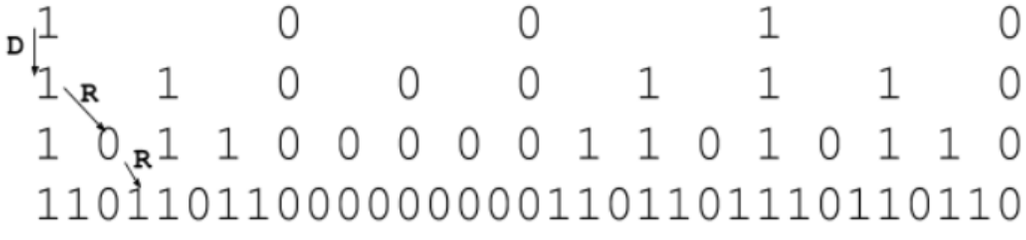
If the next command is '`D`', new $i = i+1$ (move to the next generation), new $j = 2*j$ (move exactly down according to the above table).

If the next command is '`R`', new $i = i+1$ (move to the next generation), new $j = 2*j+1$ (move right in the next generation, according to the above table).

For details please see the table below:

p T	The position (underlined) at the G-th generation denoted by 'p T' of `10010`
0 DRR	110<u>1</u>10110000000001101101110110110
2 RRR	11011011000000000110110<u>1</u>110110110
1 RDD	11011011000<u>0</u>000001101101110110110

Below the '0 `DRR`' case is shown.



Input

The first line of the input contains an integer T ($T \leq 50$) which denotes the number of test cases. T test cases follow.

There are three lines in a case. First line contains S ($1 \leq$ the length of $S \leq 100000$) and G ($1 \leq G \leq 10000$). Next two lines contain two positions in the G'th generation denoted by 'p T' ($0 \leq p <$ the length of S and the length of $T = G-1$) as above. You may assume that 'p T' is a valid input (i.e. it won't be like: $p = (|S|-1)$ and $T =$ `R...` since there is nothing to the right of the last character). You may also assume that the position denoted in the second line would not be beyond the position denoted in the third line.

Output

For each test case, output case number followed by number of '`0`'s and '`1`'s. Since number of '`0`'s and '`1`'s would be large, you need to output them by *modulo* 342307123. For details of the input/output format please look at the sample input/output.

Sample Input

```
2
10010 4
0 DRR
1 RDD
01001 2
0 D
4 D
```

Sample Output

```
Case 1: 6 4
Case 2: 4 5
```