Changing Bits

HackerRank

Let a and b be binary numbers of length n (MSB to the left). The following commands may be performed:

- `set_a idx x`: Set $a[idx]$ to x, where $0 \leq idx < n$ and $a[idx]$ is idx^{th} least significant bit of a.
- `set_b idx x`: Set $b[idx]$ to x, where $0 \leq idx < n$ and $b[idx]$ is idx^{th} least significant bit of b.
- `get_c idx`: Print $c[idx]$, where $c[idx] = a[idx] + b[idx]$ and $0 \leq idx \leq n + 1$.

Given a, b, and a list of commands, create a string made of the results of each get_c call, the only command that produces output. For example, $a = 000$ and $b = 111$ so the length of the numbers is $n = 3$. Print an answer string that contains the results of all commands on one line. A series of commands and their results follow:

```
Starting
ans = '' (empty string)
a b
000 111
set_a 1 1
010 111
set_b 0 1
010 111
get_c 3
a + b = 1001
ans = '1'
010 111
get_c 4
a + b = 01001
ans = '10'
```

Note: When the command is `get_c 4`, c had to be padded to the left with a 0 to be long enough to return a value.

Function Description

Complete the *changeBits* function in the editor below. For each `get_c` command, it should print either a `0` or a `1` without a newline until all commands have been processed. At that point, add a newline.

changeBits has the following parameters:
- *a, b*: two integers represented as binary strings
- *queries[queries[0]-queries[n-1]]*: an array of query strings in the format described

Input Format

The first line of input contains two space-separated integers, n and q, the length of the binary representations of a and b, and the number of commands, respectively.

The second and third lines each contain a string representation of a and b.
The following q lines each contain a command string $queries[i]$ as described above.

Constraints

$1 \leq n \leq 100000$
$1 \leq q \leq 500000$

Output Format

For each query of the type get_c, output a single digit 0 or 1. Output must be placed on a single line.

Sample Input 0

```
5 5
00000
11111
set_a 0 1
get_c 5
get_c 1
set_b 2 0
get_c 5
```

Sample Output 0

```
100
```

Explanation 0

- set_a 0 1 sets 00000 to 00001
- C = A + B = 00001 + 11111 = 100000, so get_c[5] = 1
- from the above computation get_c[1] = 0
- set_b 2 0 sets 11111 to 11011
- C = A + B = 00001 + 11011 = 011100, so get_c[5] = 0

The output is hence concatenation of 1, 0 and 0 = 100