B. Battle of Dust-Rain Alley

Time Limit: 5s (grader $\leq 3\text{s}$)
Memory Limit: 512MB (grader $\leq 64\text{MB}$)

This is an interactive problem. Submissions must be in C++ (version no lower than C++14).

Problem Description

In the ancient legend of the Flea Kingdom, the Dust-Rain Alley is a mysterious circular alley shrouded in mist.
Each rainy season, dust and fog interweave like a curtain, and the n magical lamps in the alley appear and disappear.

As the most trusted explorer of King Volt, you are teleported to a magical lamp—you know neither the length n of the alley, nor your starting position, nor the states of the lamps.

Before the rainy season ends, you must complete King Volt's task: determine the length n of the alley!

Dust-Rain Alley is a circular alley of length n. Along it are n magical lamps, each either on or off.
At the start, you're teleported to one lamp, but you don't know which one or the states of any lamps.

To discover n, you can cast several spells, each one of the four types below:

- **Rain-veil Traverse (CW)**: move clockwise to the next lamp, consuming 1 stamina.
- **Rain-veil Traverse (CCW)**: move counter-clockwise to the next lamp, consuming 1 stamina.
- **Dust's Glimpse**: query the state of the current lamp, consuming 1 stamina.
- **Fog Ritual**: toggle the state of the current lamp (on→off or off→on), costing **no stamina**.

You have limited time—you must finish before the rainy season ends (at most 2×10^6 spell casts) and report the length n. Additionally, King Volt will reward you based on stamina used, as detailed in the "Subtasks" section.

Implementation Details

You should NOT implement `main`.

Include the provided header:

```
#include "lane.h"
```

Implement the function:

```
int solve(int c);
```

- `c` is the subtask ID given by King Volt.
- The function should return n, the length of the alley.
- Your program **may be called multiple times per test case**.

Use these functions to interact:

```
void clockwise();        // CW move, costs 1 stamina
void anticlockwise();    // CCW move, costs 1 stamina
bool ask();              // query current lamp, costs 1 stamina; returns true if on, false if of
void flip();             // toggle current lamp, costs 0 stamina
```

Testing Locally

The provided `grader.cpp` is a reference; the actual grader is different, adaptive, may change n or lamp states dynamically (as long as consistent with past `ask` results).

Compile with:

```
g++ grader.cpp lane.cpp -o lane -O2 -std=c++14 -static
```

For debug mode add `-DDEBUG`.

```
g++ grader.cpp lane.cpp -o lane -O2 -std=c++14 -static -DDEBUG
```

The grader reads from stdin:

- First line: `c, T` (subtask ID and number of testcases).
- Each testcase: a line with n, then a line with an n-character string of `0` / `1` (initial states).

Your `solve` will be called for each testcase. After all testcases, the grader outputs your score and evaluation.

- The first line will show your score.
- The second line will display a description of the test results from the interaction library.

In DEBUG mode it logs detailed interaction to stderr. Please use small test data to avoid huge logs.

Example Interaction

Suppose $n = 2$, initial states `[0,1]` :

Your Code	Grader	Notes
—	`solve(2)`	start
`ask()`	returns `0`	position 1 (off), −1 stamina
`clockwise()`	—	move to pos 2, −1 stamina
`ask()`	returns `1`	position 2 (on), −1
`clockwise()`	—	move to pos 1, −1
`ask()`	returns `0`	position 1 (off), −1
`flip()`	—	toggle to on, free
`anticlockwise()`	—	move to pos 2, −1
`anticlockwise()`	—	move to pos 1, −1
`ask()`	returns `1`	position 1 (on), −1
return `2`	—	correct, 9 spells, 8 stamina

Files Included

- `grader.cpp` : reference interactive grader
- `lane.h` : header
- `template_lane.cpp` : sample code
- `lane1.in` , `lane2.in` , `lane3.in` : large samples for subtasks

Subtasks

Let T be total calls to `solve` , and $\sum n$ total alley lengths. Constraints: $1 \le n \le 10^5$, $1 \le T \le 10^6$, $\sum n \le 10^7$.

Volt issues 3 subtasks:

1. **First Survey** (subtask 1, 3 pts):
   - $T \le 10$, **all lamps initially off**.
   - Any correct method using ≤ 2×10^6 spells per testcase gets full points.

2. **Misty Labyrinths** (subtask 2, 40 pts):
    - $n \le 2000, T \le 5000$.
    - ≤ $20n$ spells per testcase.
    - Let $x = \max(q/n)$ over testcases, where q is stamina used. Score:
        - $x \le 7.83$: 40 pts (full)
        - $x > 15$: 0 pts
        - else: $40 \times \frac{15-x}{15-7.83}$.
3. **Final Challenges** (subtask 3, 57 pts):
    - No special limits.
    - ≤ $20n$ spells per testcase.
    - Satisfaction per testcase:
        - if $n \le 2000$: $q/(2n)$
        - else: q/n
    - Let $x = \max$ satisfaction. Score:
        - $x \le 5.35$: 57 pts (full)
        - $x > 8$: 0 pts
        - else: $57 \times \frac{8-x}{8-5.35}$.

Whole-subtask score: min over all testcases, floored to two decimals.

Tips

- Don't illegally read lamp states or poke stdin/stdout—cheating.
- **The actual grader may adapt n and lamp states, as long as consistent.**

Candidates must also follow the standard constraints as in traditional problems. For example, a compilation error causes a score of 0 for the entire problem; a runtime error, time limit exceeded, or memory limit exceeded will cause a score of 0 for the corresponding test case. You are only permitted to access variables or data you define yourself or that are provided by the interaction library, and their associated memory spaces. Attempting to access any other memory location may result in compilation or runtime errors.