Fortieth Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	Filename
Randomized Decisions	randomized
Conveyor Confuse-inator	conveyor
Take a Pancake	pancake
Wormhole Wizardry	wormhole
Pirates on a Boat!	pirates
Toby Talks	talk
Displaying Gazebos	displaying
Inexplicable Indecision	indecision
Deluxe Defense	defense
Elephant Escape Artists	escapeartists
Stonk Trading	stonks
Careening Crates	crates

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py
Call your Java class: *filename*

For example, if you are solving Toby Talks:
Call your program file: talk.c, talk.cpp, talk.java, or talk.py
Call your Java class: talk

Notes

1. All solutions must read from standard input and write to standard output. The judges will ignore all output sent to standard error.

2. All input sets used by the judges will follow the input specification in each problem. You do not need to test for input that violates the input format specification.

3. Unless otherwise specified, specification for input and output that refers to "uppercase letters" means letters 'A' to 'Z' (inclusive) as they are used in English; input and output that refers to "lowercase letters" means 'a' to 'z' (inclusive) as they are used in English.

4. Submitted programs will be tested on additional judge cases and not just the sample cases given in the problem.

5. If you need to use the value for π, use the value 3.141592653589793.

6. If you want to know how to compute the absolute error, given the actual value of a quantity, x, and the measured value of a quantity, x_0, then the absolute error value, Δx, is calculated as:

$$\Delta x = |x_0 - x|$$

7. The relative error is defined as the ratio of the absolute error of the measurement to the actual measurement. If you want to know how to compute the relative error, given the actual value of a quantity, x, and the measured value of a quantity, x_0, then the relative error value, r, is calculated as:

$$r = \frac{|x_0 - x|}{x} = \frac{\Delta x}{x}$$

8. The modulo operation (often designated with the operator %) is defined as the remainder obtained after two operands are divided (e.g. 11 % 2 = 1, 29 % 3 = 2).

9. Images used are owned by either the original author, the University of Central Florida, or used under appropriate licenses (such as various Creative Commons licenses).

Randomized Decisions

Filename: `randomized`

Decisions can be difficult, especially under a strict deadline! Due to a modicum of procrastination, Aga has now ended up with a lot of tasks she needs to do by 11:59 pm.

She has two types of tasks: solving competitive programming problems and completing homework assignments. To decide between solving a problem or completing an assignment, Aga has the idea to let a coin flip decide. If she flips heads, she'll solve a competitive programming problem; if she flips tails, she'll do a homework assignment.

Unfortunately, randomness is unpredictable. Even with a perfectly weighted coin, it is possible that a series of coin flips will not give her the right number of programming problems to solve. This is, of course, unacceptable, so instead of relying on the uncertainty of coin flips, Aga has asked you to create a list of coin flip results that she can use to complete all her tasks.

The Problem:

Given the total number of tasks and the number of tasks which are competitive programming problems, generate a list of coin flips to determine an order of the problems and the homework assignments.

The Input:

The only line of input contains two integers, n and m ($1 \leq n \leq 1{,}000$; $0 \leq m \leq n$), representing the total number of tasks Aga needs to do and the number of problems Aga wants to solve, respectively.

The Output:

Output n lines, m of which contain only the string "`Heads!`" and the rest of which contain only the string "`Tails!`" If there are multiple correct solutions, any of them will be accepted.

Sample Input 1:

```
4 2
```

Sample Output 1:

```
Heads!
Tails!
Tails!
Heads!
```

Sample Input 2:

```
3 0
```

Sample Output 2:

```
Tails!
Tails!
Tails!
```

Conveyor Confuse-inator

Filename: `conveyor`

Dr. Hunts Mittleshmirtz has created another dastardly contraption to finally trap Agent B: the Conveyor Confuse-inator! Agent B knows he might eventually be trapped in the machine, and as a result he is planning multiple escape routes.

The machine consists of a giant grid of conveyors going in various directions. Every cell has a conveyor, each one pointing either directly up ('^'), down ('v'), left ('<'), or right ('>'). In other words, each conveyor points to some cell that is cardinally adjacent to it. When Agent B is on a conveyor, it will push him to the cell that the conveyor points to.

Luckily, Agent B managed to obtain the machine's layout, but he doesn't know onto which part of the machine Mittleshmirtz will drop him. For various starting locations, Agent B has found rectangular regions such that, if the conveyors take him strictly outside them, he has an escape route. Otherwise, Agent B may be trapped inside of the machine forever!

Note that any conveyor that points outside the grid will allow Agent B to escape automatically.

The Problem:

Given the grid of conveyors, answer queries of the following form: if Agent B starts at the cell (r, c), determine if he will ever travel outside a given bounding rectangle.

The Input:

The first line of input contains three integers, n, m and q ($1 \le n \le 500$; $1 \le m \le 500$; $1 \le q \le 10^5$), representing the number of rows, the number of columns and the number of queries, respectively. The next n lines each contain m characters, each either '^', 'v', '<', or '>', representing the direction of each cell's conveyor.

The next q lines each contain 6 integers, representing an individual query: r_s, c_s, r_1, c_1, r_2, and c_2 ($1 \le r_1 \le r_s \le r_2 \le n$; $1 \le c_1 \le c_s \le c_2 \le m$). A coordinate pair (r, c) corresponds to the grid cell in the r^{th} row from the top and the c^{th} column from the left. Coordinate (r_s, c_s) corresponds to Agent B's starting cell, and (r_1, c_1) and (r_2, c_2) denote the inclusive top-left and bottom-right corners of the rectangle.

The Output:

For each of the q queries, output "`YES`" if Agent B ever escapes the bounding rectangle, or "`NO`" otherwise.

(Sample Input and Sample Output on next page)

Sample Input 1:

```
5 5 3
<<<v<
>>^<<
v>v<<
v<>>^
^^^^^
1 5 1 4 2 5
3 1 1 1 5 2
3 4 3 3 4 5
```

Sample Output 1:

```
YES
NO
NO
```

Sample Input 2:

```
2 2 3
<>
><
1 1 1 1 1 1
2 2 2 2 2 2
2 1 1 1 2 2
```

Sample Output 2:

```
YES
YES
NO
```

Take a Pancake

Filename: `pancake`

Kevin and Jakob have just created the Highly Stacked Pancake Tower (HSPT), a delicious dish composed of n pancakes arranged in a single column. They want to submit their masterpiece to a culinary competition, but the rules of the competition allow only one chef to take credit for the entry. To determine who will claim credit for creating the HSPT, Kevin and Jakob have agreed to play a game.

The game starts with a single tower containing n pancakes. Kevin and Jakob alternate turns, beginning with Kevin. On their turn, a player must remove a positive integer number of pancakes from the tower that does not exceed half of the number of pancakes currently in the tower (rounded up). For instance, if there are currently 5 pancakes in the tower, a player can choose to remove 1, 2, or 3 pancakes, or if there are 4 pancakes, a player could remove 1 or 2 of them. The chosen pancakes are permanently removed from the tower, and the player who removes the final pancake wins the game and the rights to the HSPT.

As an aspiring chef, Jakob wants to further his career by taking credit for the HSPT. To this end, he has decided to stack the odds in his favor by adding some number of pancakes (possibly none) on top of the tower before the game begins to ensure that he wins. Since making pancakes can be time-consuming, Jakob has asked you to find the minimum number of pancakes that he must add to the top of the HSPT to guarantee his victory.

The Problem:

Given the number of pancakes initially in the tower, determine the minimum number of pancakes Jakob must add to guarantee he will win the game, assuming both players play optimally.

The Input:

The input contains a single integer, n ($1 \leq n \leq 10^9$), representing the initial number of pancakes in the tower.

The Output:

Output a single integer: the minimum number of pancakes that Jakob must add on top of the tower to win assuming both players play optimally.

Sample Input 1:	**Sample Output 1:**
1	1

Sample Input 2:	**Sample Output 2:**
9	5

Sample Input 3:	**Sample Output 3:**
30	0

Wormhole Wizardry

Filename: `wormhole`

The world-renowned wormhole wizard, Wesley W. Wormwood, has wandered his way into the Unwinding Wilds: a one-dimensional land full of wonder and whimsy. In the Unwinding Wilds, there are n cities at integer positions 1 through n on the number line. The uninhabited parts of the Wilds are wayward and wicked, making travel between these cities unwise.

Ever the warmhearted wizard, Wesley has decided to solve this problem for the locals. He will open $n - 1$ wormholes, each of which allow two-way travel between two cities, such that each city can reach every other city through some sequence of wormholes. However, the Unwinding Wilds have strange effects on Wesley's magics: for some constant k, Wesley can only open a wormhole between two cities if their positions on the number line differ by no more than k. Furthermore, each wormhole spell Wesley casts depletes some of his Will: if he opens a wormhole between cities at positions i and j, the amount of Will he must spend is equal to $i + j$.

Unfortunately, Wesley has not been able to figure out the value of this constant k. With this in mind, for all possible values of k, Wesley would like to determine the minimum amount of Will he must use to establish his wormhole network.

The Problem:

For each value of k from 1 to $n - 1$ (inclusive), determine the minimum cost required to open $n - 1$ wormholes such that each city can reach any other through the wormhole network. A wormhole can be opened between cities at positions i and j only if $|i - j| \leq k$, and the cost of opening such a wormhole is equal to $i + j$.

The Input:

The only line of input contains a single integer, n ($2 \leq n \leq 10^5$), representing the number of cities in the Unwinding Wilds.

The Output:

Output $n - 1$ lines. On the k^{th} line, output a single integer: the minimum cost of Wesley's wormhole network for that value of k. It can be shown that a valid network always exists.

Sample Input 1:

```
2
```

Sample Output 1:

```
3
```

Sample Input 2:

```
4
```

Sample Output 2:

```
15
13
12
```

Pirates on a Boat!

Filename: `pirates`

Recently, a map to the fabled Isle of Impossibly Invaluable Treasure in the Serpent Seas has been uncovered. Captain Trell wants to retrieve this impossibly invaluable treasure and is getting ready to sail.

He'd like to set sail as soon as possible, but the Serpent Seas are dangerous, so Captain Trell needs his entire crew on board to make it to the treasure. Luckily, he knows the positions of his rectangular axis-aligned boat (meaning, the sides of the boat are parallel to the vertical or horizontal axis) and the positions of all his crew. Now all that's left to do is to determine if all the pirates are on the boat!

The Problem:

Given the positions of two opposite corners of a rectangular axis-aligned boat, and the positions of the members of Captain Trell's pirate crew, determine if every pirate is on the boat.

The Input:

The first line contains four integers, x_1, y_1, x_2 and y_2 ($0 \le x_1 < x_2 \le 100$; $0 \le y_1 < y_2 \le 100$), representing the x and y coordinates of the lower-left and upper-right corners of the boat, respectively.

The next line contains a single integer, n ($1 \le n \le 10$), representing the number of pirates in Captain Trell's crew. The i^{th} of the following n lines contains two integers, x_i and y_i ($0 \le x_i \le 100$; $0 \le y_i \le 100$), indicating that the i^{th} pirate is at the coordinate (x_i, y_i). A pirate is on the boat if they touch any part of it, including the interior, an edge, or a corner.

The Output:

If all the pirates are on the boat, output "`A boat full of pirates!`" Otherwise, output "`Pirate overboard!`"

Sample Input 1:

```
0 0 5 5
3
3 4
2 6
5 0
```

Sample Output 1:

```
Pirate overboard!
```

Sample Input 2:

```
1 2 3 3
2
1 3
3 2
```

Sample Output 2:

```
A boat full of pirates!
```

Toby Talks

Filename: `talk`

Toby is quite the talkative cat. He has so much to say that his owner can't keep count! Please help Toby make his voice heard by counting the number of times he meows.

When Toby gets frustrated, he might say something like "mrow" or "meroww." This is unprofessional of him, and he has kindly asked you to keep this off the record.

A proper and professional "meow" is counted if and only if it is complete and uninterrupted; that is, if the letters '`m`', '`e`', '`o`', and '`w`' occur consecutively, in that order, with no other letters in between them.

The Problem:

Given a string, determine how many times a proper and professional "meow" occurs.

The Input:

The input contains a single string of lowercase letters. The length of this string is between 1 and 500 characters, inclusive, and is guaranteed to contain only lowercase English letters with no whitespace.

The Output:

Output a single integer: the number of occurrences of "meow" in the given string.

Sample Input 1:	**Sample Output 1:**
`meowmeow`	`2`

Sample Input 2:	**Sample Output 2:**
`meowwwwwwww`	`1`

Sample Input 3:	**Sample Output 3:**
`merow`	`0`

Sample Input 4:	**Sample Output 4:**
`meowhissmeow`	`2`

Displaying Gazebos

Filename: `displaying`

Thomas has been trying to show more people his collection of gazebos! Unfortunately, many of his friends have gotten tired of his fascination with them. To combat this, Thomas has decided to create a gazebo display that will impress even the greatest critics of gazebos.

Thomas has a line of n gazebos. Each gazebo is either round (denoted by '`R`') or square (denoted by '`S`'). Thomas would like his final arrangement to satisfy two criteria:

- No two adjacent gazebos should be the same shape (to prevent monotony).
- Thomas also loves palindromes, so the final arrangement should be the same when the order is reversed.

To create his display, he can choose to remove some gazebos from the line and push together those that remain. Can you help Thomas find the minimum number of gazebos he must remove from the line to create an acceptable gazebo display?

The Problem:

Given the shape of each gazebo in the line, determine the minimum number of gazebos Thomas has to remove from the line so that the gazebo display satisfies both of his criteria.

The Input:

The first line of input contains a single integer, n ($1 \leq n \leq 10^5$), representing the number of gazebos. The next line contains a string of n characters, the i^{th} of which denotes the shape of the i^{th} gazebo. It is guaranteed that the string only contains the characters '`R`' and '`S`'.

The Output:

Output a single integer value: the minimum number of gazebos Thomas must remove from the line to create his arrangement.

Sample Input 1:

```
7
RSSSRRS
```

Sample Output 1:

```
4
```

Sample Input 2:

```
6
SRRRRS
```

Sample Output 2:

```
3
```

Inexplicable Indecision

Filename: `indecision`

Tyler is putting together a problem set! He has a potential ordering of all the problems he will include, as well as their associated difficulties. However, Tyler would like to make the problem set perfect, and he's worried that the current ordering might not be optimal.

Tyler keeps looking at the current order of problems and thinking that an adjacent pair of problems might be better in the opposite order. To test this, he will perform a series of swaps between adjacent problems. Throughout this process, Tyler occasionally wants to know the sum of difficulties of certain ranges of problems so that he can inform his decisions on the ordering of the set. That is, for certain [l, r] ranges, he would like to know the sum of problem difficulties at all positions between l and r, inclusive.

Unfortunately, Tyler can't figure out how to efficiently compute these sums of difficulties. Can you help him by finding these sums while performing the requested swap operations?

The Problem:

Given an array representing the initial order of difficulty values in Tyler's problem set, perform adjacent swap operations and answer queries on the sums of difficulties on contiguous ranges.

The Input:

The first line of input contains two integers, n ($2 \le n \le 10^5$) and m ($1 \le m \le 10^5$), representing the number of problems and the total number of operations, respectively. The following line contains n space-separated integers, the i^{th} of which represents the difficulty of the i^{th} problem. Each problem difficulty is between 1 and 10^4, inclusive.

The following m lines describe the requested operations for you to perform. Each of these lines begins with a string, either "`swap`" or "`sum`". If the string is "`swap`", it will be followed by a single integer, i ($1 \le i < n$), representing that the problems at positions i and $i + 1$ will be swapped. If the string is "`sum`", it will be followed by two integers, l and r ($1 \le l \le r \le n$), representing a request for the sum of problem difficulties between positions l and r, inclusive.

The Output:

For each "`sum`" operation, output a single integer on its own line: the sum of problem difficulties between positions l and r (inclusive) after performing all previous swap operations.

(Sample Input and Sample Output on next page)

Sample Input 1:

```
5 4
1 3 5 2 4
sum 1 5
sum 1 3
swap 3
sum 1 3
```

Sample Output 1:

```
15
9
6
```

Sample Input 2:

```
4 4
1 3 2 3
sum 3 4
swap 1
swap 2
sum 3 4
```

Sample Output 2:

```
5
4
```

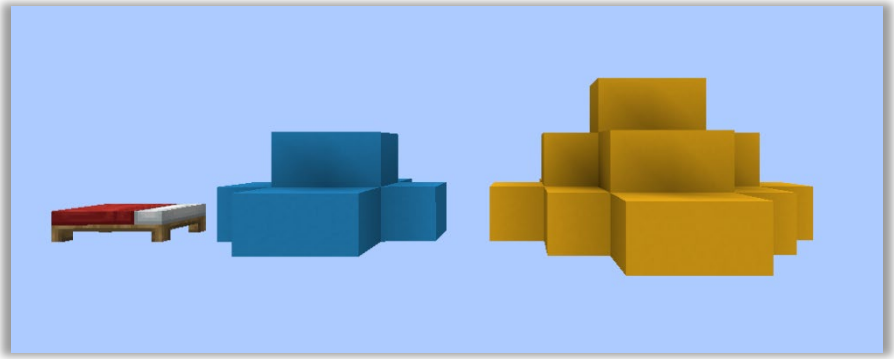
Deluxe Defense

Filename: `defense`

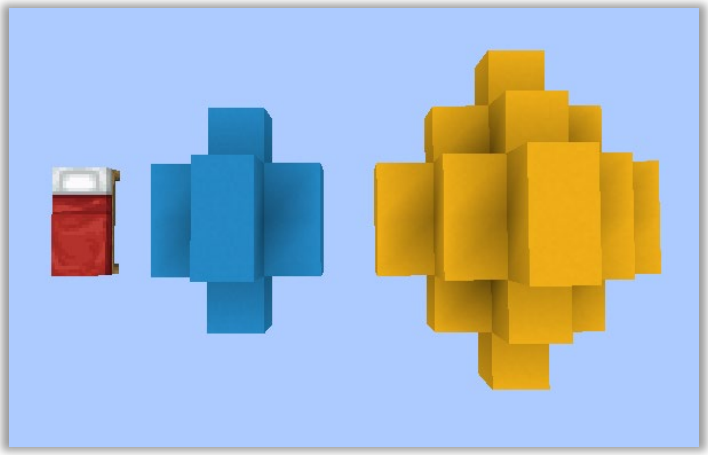
An important part of Bedwars, the Minecraft minigame, is protecting your team's bed—an object 2 blocks long, 1 block wide, and 1 block tall—with a bed defense. Players do this by placing layers of "unit" blocks (each measuring 1 unit long, 1 unit wide, and 1 unit tall) onto and around their bed, where the innermost layer uses the minimum number of these blocks to completely cover the bed with each subsequent layer completely covering the previous one.

Formally, if a bed defense has x layers and y is the minimum number of blocks that must be placed such that no part of the x^{th} layer is exposed, then the $(x + 1)^{\text{th}}$ layer of the bed defense requires y blocks. Blocks cannot be placed anywhere below the bed.

Caden loves playing Bedwars, and while waiting for his next game to queue, he wonders how many blocks are needed to build the *outermost* layer of some bed defense, not including any previous layers. Can you help him with this?



Side view of bed defenses with 0, 1, and 2 layers, respectively



Top view of bed defenses with 0, 1, and 2 layers, respectively

The Problem:

Given the number of layers in a bed defense, determine the minimum number of blocks required to build the outermost layer.

The Input:

The only line of input contains a single integer, k ($1 \leq k \leq 10^9$), representing the number of layers.

The Output:

Output a single integer value: the minimum number of blocks required to build the k^{th} layer of a bed defense with k layers. Since the answer can be large, output it modulo $10^9 + 7$.

Sample Input 1:	**Sample Output 1:**
1	8

Sample Input 2:	**Sample Output 2:**
3	32

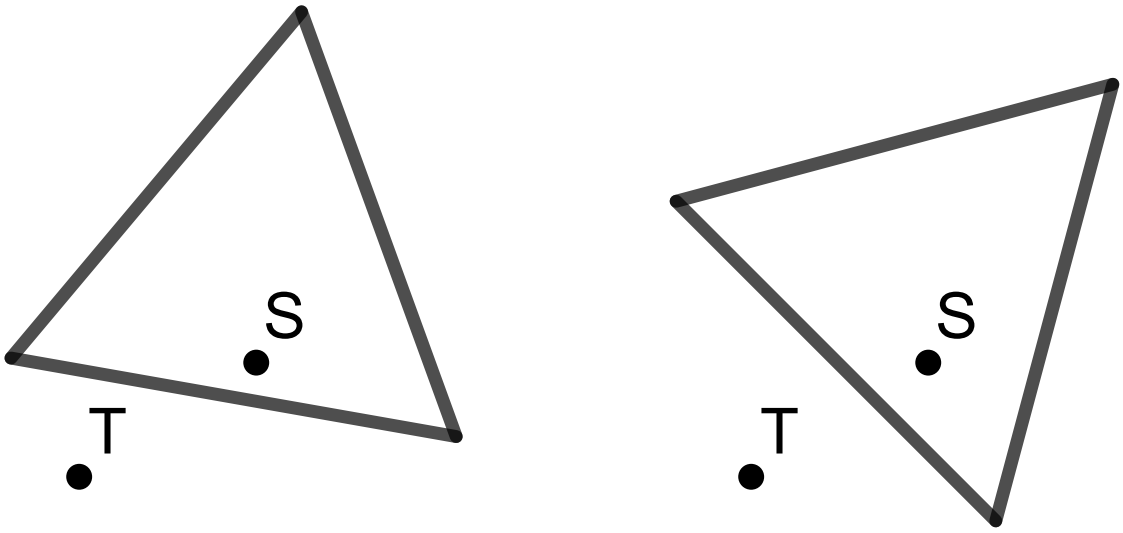
Sample Input 3:	**Sample Output 3:**
7495610	367740066

Elephant Escape Artists

Filename: `escapeartists`

In the progression of the Great Zoo Wars, the zookeepers successfully captured some of the elephants' most elite spies: Elliot, Ellie, and Ellanah. The zookeepers, worried about the elephants' ability to escape, have caged the elephants in an electrified fence that takes the shape of a regular polygon. They still don't think that is secure enough, so they have decided to make the fence continuously rotate about the center of the polygon. A regular polygon is a polygon where all of its sides are equal in length, and all of its interior angles are equal in measure. The center of a regular polygon is the point that is equidistant to all vertices of the regular polygon.

Luckily for the spies, with Ellanah also being captured, the elephants have the ability to craft some quite complex technology, including her patent-pending Elephant Elevation Engine! To use this piece of technology, they must choose one point inside the cage and another point outside of the cage, which will allow them to be transported across the fence. Since Ellanah is working with limited resources, she needs to minimize the distance between the two chosen points. However, she cannot risk the rotating electric fence colliding with the engine. That is, the two points that are chosen must always remain on opposite sides of the fence, regardless of how much the fence rotates.



The images above depict two rotations of a regular triangle where point *S* is always inside the triangle and *T* is always outside the triangle.

The Problem:

Given the dimensions of a regular polygonal fence rotating about its center, determine the minimum distance between the two chosen points for the Elephant Elevation Engine, such that one point is always inside the enclosure and the other point is always outside the enclosure.

The Input:

The only line of input consists of two integers, n and s ($3 \leq n \leq 10^5$; $1 \leq s \leq 10^4$), representing the number of sides of the regular polygon and the length of each side of the regular polygon, respectively.

The Output:

Output a single real number: the minimum distance between the two chosen points required for the Elephant Elevation Engine to function. Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Sample Input 1:	**Sample Output 1:**
3 10	2.8867513459

Sample Input 2:	**Sample Output 2:**
4 25	5.1776695296

Stonk Trading

Filename: `stonks`

Gian is a competitive programmer and aspiring day trader. He is working on an advanced algorithm to trade stocks automatically. However, there's a key part of the algorithm he hasn't completed yet: Determining whether a stock hit a certain price during a specific time period.

Gian has already recorded the stock price at several arbitrary points in time. Using this information, he wants to answer questions of the following form: Did the stock price reach a specific price, p, between two times t_1 and t_2, inclusive?

The stock price is modeled as a continuous function. A continuous function is a function whose graph has no breaks, holes, or jumps, allowing it to be drawn on a paper without lifting a pencil.

Since Gian doesn't have data for every single moment in time, and the stock price can change continuously between the recorded points, it's always possible that the stock hit the price p. But Gian is only interested in cases where he can be *certain* that the stock price reached p using only the information that he has. Gian is certain that the stock price reached p if there is no continuous stock behavior that follows all of the recorded data and avoids hitting the price p between times t_1 and t_2, inclusive.

The Problem:

Given the price of a stock at certain points in time, determine, with *certainty*, if the stock price hit the specified price p between two given times t_1 and t_2.

The Input:

The first line contains a single integer, n ($1 \le n \le 10^5$), representing the number of data points that Gian has for the stock price. The next n lines each contain two integers, t_i ($1 \le t_i \le 10^5$) and c_i ($1 \le c_i \le 100$), indicating that at time t_i the stock had price c_i. It is guaranteed that the data points are given in increasing order of time and that no two data points share the same time.

The next line of input contains a single integer, q ($1 \le q \le 10^5$), indicating the number of questions Gian needs to answer for his algorithm. The next q lines of input each contain 3 integers, p_j, s_j and e_j ($1 \le p_j \le 100$; $1 \le s_j \le e_j \le 10^5$), representing the price, the interval start time and the end time, respectively, for the j^{th} question.

The Output:

Output q lines of output, where the i^{th} line contains "`Stonks!`" if the conditions of the i^{th} question are guaranteed and "`Not Necessarily Stonks!`" otherwise.

(Sample Input and Sample Output on next page)

Sample Input 1:

```
3
1 5
3 6
5 4
5
6 1 1
5 1 1
6 1 3
5 2 4
6 2 4
```

Sample Output 1:

```
Not Necessarily Stonks!
Stonks!
Stonks!
Not Necessarily Stonks!
Stonks!
```

Sample Input 2:

```
7
1 2
3 5
4 3
5 8
6 1
7 10
8 3
4
7 1 4
7 3 6
12 1 10
3 7 8
```

Sample Output 2:

```
Not Necessarily Stonks!
Stonks!
Not Necessarily Stonks!
Stonks!
```

Careening Crates

Filename: `crates`

Oliver is an employee for the shipping company "High Speed Package Transportation (HSPT)." He has just finished loading the next collection of crates onto a flatbed truck when he realizes that he forgot to tie them down! Oliver always loads the crates in a single layer, but now the crates might slide around due to the abrupt movement of the truck!

When the truck accelerates, the crates slide backward. When it brakes, the crates slide forward. When it turns left, the crates slide right. When it turns right, the crates slide left. When the crates slide, all of them start moving at once, and they will continue moving in the given direction until the crates are stopped by either a wall or by colliding with an already stopped crate. Sliding happens instantaneously to all crates, finishes before any subsequent movement, and happens independently of any previous movement.

A crate will slide until it collides with either a wall of the truck bed or another crate. Oliver has access to a complete list of sudden motions the truck will make, in order. Help him figure out the final positions of all the crates so that crane operators will be ready to unload them when they arrive.

The Problem:

Given a square grid representing the initial state of the truck bed and a list of sequential motions the flatbed truck will make, compute the locations of the crates at the end of the trip.

The Input:

The first line of input contains a single integer, n ($1 \leq n \leq 1{,}000$), representing the side length of the n by n truck bed. After this, there are n lines, each containing n characters, describing the truck bed in the form of a grid. Each character is either '`.`' (representing an empty space) or '`#`' (representing a crate).

The next line contains a single integer, k ($1 \leq k \leq 10^5$), representing the number of motions of the truck. The next k lines describe the truck's motions, each containing exactly one of four strings: "`ACCELERATE`", "`BRAKE`", "`LEFT`", or "`RIGHT`".

The Output:

Output an n by n grid of characters representing the final arrangement of the crates on the truck bed. Each character should either be '`.`' (empty space) or '`#`' (crate).

(Sample Input and Sample Output on next page)

Sample Input 1:

```
3
#..
.#.
...
1
ACCELERATE
```

Sample Output 1:

```
...
...
##.
```

Sample Input 2:

```
4
##..
#...
...#
.#..
5
BRAKE
LEFT
ACCELERATE
RIGHT
BRAKE
```

Sample Output 2:

```
###.
##..
....
....
```