Task Overview Sheet

TASK		GUESS	ROBOTS	BOUNDARY
Time limit per test case		1 second CPU	2 seconds CPU	1 second CPU
Memory limit		64 MB	64 MB	64 MB
Compiler options	**C**	`-pipe -O2 -lm`		
	C++	`-pipe -include /usr/include/stdlib.h -O2 -lm`		
	Pascal	`-So -O1 -XS`		
Number of tests		20	20	25
Maximum points per test		5	5	4
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: guess`
		`LANG: C`
		`*/`
			`/*`
			`TASK: robots`
			`LANG: C`
			`*/`
				`/*`
				`TASK: boundary`
				`LANG: C`
				`*/`

Program header comment when using C++		`/*`
		`TASK: guess`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: robots`
			`LANG: C++`
			`*/`
				`/*`
				`TASK: boundary`
				`LANG: C++`
				`*/`

Program header comment when using Pascal		`{`
		`TASK: guess`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: robots`
			`LANG: PASCAL`
			`}`
				`{`
				`TASK: boundary`
				`LANG: PASCAL`
				`}`

Solution acceptance rule		Example input is solved correctly.	Example input is solved correctly.	Example input is solved correctly.

Guess Which Cow (interactive task)

TASK

The N ($1 \le N \le 50$) cows in Farmer John's herd look very much alike and are numbered 1…N. When Farmer John puts a cow to bed in her stall, he must determine which cow he is putting to bed so he can put her in the correct stall.

Cows are distinguished using P ($1 \le P \le 8$) properties, numbered 1…P, each of which has three possible values. For example, the color of a cow's ear tag might be yellow, green, or red. For simplicity, the values of every property are represented as the letters '`X`', '`Y`', and '`Z`'. Any pair of Farmer John's cows will differ in at least one property.

Write a program that, given the properties of the cows in Farmer John's herd, helps Farmer John determine which cow he is putting to bed. Your program can ask Farmer John no more than 100 questions of the form: Is the cow's value for some property T in some set S? Try to ask as few questions as possible to determine the cow.

Input: `guess.in`

- The first line of the input file contains two space-separated integers, N and P.
- Each of the next N lines describes a cow's properties using P space-separated letters. The first letter on each line is the value of property 1, and so on. The second line in the input file describes cow 1, the third line describes cow 2, etc.

Example input:

```
4 2
X Z
X Y
Y X
Y Y
```

Interactivity: *standard input and output*

The question/answer phase takes place via standard input and standard output.

Your program asks a question about the cow being put to bed by writing to standard output a line that is a '`Q`' followed by a space, the property number, a space, and a space-separated set of one or more values. For example, "`Q 1 Z Y`" means "Does property 1 have value either '`Z`' or '`Y`' for the cow being put to bed?". The property must be an integer in the range 1…P. All values must be '`X`', '`Y`', or '`Z`' and no value should be listed more than once for a single question.

After asking each question your program asks, read a single line containing a single integer. The integer 1 means the value of the specified property of the cow being put to bed is in the set of values given; the integer 0 means it is not.

The program's last line of output should be a '`C`' followed by a space and a single integer that specifies the cow that your program has determined Farmer John is putting to bed.

Example exchange (for example input above):

Input	*Output*	*Explanation*
	`Q 1 X Z`	
`0`		Could be cow 3 or cow 4.
	`Q 2 Y`	
`1`		Must be cow 4!
	`C 4`	
program exits		

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

Correctness: 30% of points

Programs will receive full score on correctness only if the cow specified is the only cow that is consistent with the answers given. A program that asks more than 100 questions for a test case will receive no points for that test case.

Question count: 70% of points

The remaining points will be determined by the number of questions required to correctly determine the cow. The test cases are designed to reward minimizing the worst-case question count. Partial credit will be given for near-optimal question counts.

Amazing Robots

TASK

You are the proud owner of two robots that are located in separate rectangular mazes. Square (1, 1) in a maze is the square in the upper-left corner, which is the north-west corner. Maze i (i = 1, 2) has a set of G_i ($0 \le G_i \le 10$) guards trying to capture the robots by patrolling back and forth on a straight patrol path. Your goal is to determine a sequence of commands such that the robots exit the mazes without either robot being captured by a guard.

At the beginning of each minute, you broadcast the same command to both robots. Each command is a direction (north, south, east, or west). A robot moves one square in the direction of the command, unless the robot would collide with a wall, in which case the robot does not move for that minute. A robot exits the maze by walking out of it. A robot ignores commands after it has exited its maze.

Guards move one square at the beginning of each minute, at the same time as the robots. Each guard begins at a given square facing a given direction and moves forward one square per minute until the guard has moved one fewer square than the number of squares in his patrol path. The guard then turns around instantaneously and walks in the opposite direction back to his starting square, where he turns around again and repeats his patrol path until each robot has exited its maze.

A guard's patrol will not require the guard to walk through walls or exit the maze. Although guard patrols may overlap, no two guards will ever collide: they will never occupy the same square at the end of a minute, and they will not exchange squares with each other during a minute. A guard in a maze will not start in the same square as the robot in that maze.

A guard captures a robot if the guard occupies the same square at the end of a minute as the robot, or if the guard and the robot exchange squares with each other during a minute.

Given two mazes (each no larger than 20x20) along with the initial square of each robot and the patrol paths of the guards in each maze, determine a sequence of commands for which the robots exit without being caught by the guards. Minimize the time it takes for the later robot to exit its maze. If the robots exit at different times, the time at which the earlier robot exited does not matter.

Input: `robots.in`

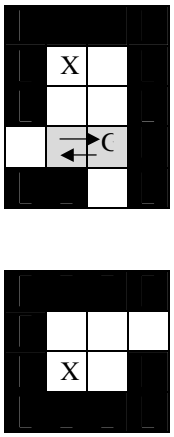
The first set of lines describes the first maze and its occupants. Subsequently, the second set of lines describes the second maze and its occupants.

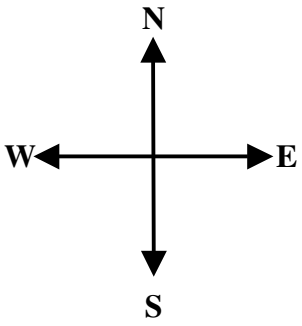
- The first line of input contains two space-separated integers, R_1 and C_1, the number of rows and columns in maze 1.
- The next R_1 lines each contain C_1 characters specifying the maze layout. The robot's starting square is specified by an '`X`'. A '`.`' represents an open space and '#' represents a wall. Each maze will contain exactly one robot.
- Following the maze layout is a single line with a single integer G_1, the number of guards in the first maze ($0 \le G_1 \le 10$).
- Each of the next G_1 lines describes a guard's patrol as three integers and a character, separated by single spaces. The first two integers specify the row and column of the starting square of the guard. The third integer specifies the number of squares (2…4) in the guard's patrol path. The character specifies the initial direction the guard is facing: '`N`', '`S`', '`E`', or '`W`' (north, south, east, and west, respectively).

The description of the second maze follows the description of the first, in the same format but with potentially different values.

Example input:

```
5 4
####
#X.#
#..#
...#
##.#
1
4 3 2 W
4 4
####
#...
#X.#
####
0
```





Output: `robots.out`

The first line of the output should contain a single integer K ($K \le 10000$), the number of commands for both robots to exit the maze without being captured. If such a sequence of commands exists, the shortest sequence will have no more than 10000 commands. The next K lines are the sequence of commands, each containing a single character from the set {'`N`', '`S`', '`E`', '`W`'}. If no such sequence exists, output a single line containing "`-1`".

Both robots should exit their mazes by the end of the commands. The last command should cause at least one of the robots to exit its maze. If multiple sequences of commands cause the robots to exit in the minimum time, any will be accepted.

Example output:

```
8
E
N
E
S
S
S
E
S
```

CONSTRAINTS

Running time	2 seconds of CPU
Memory	64 MB

SCORING

No partial credit will be given on test cases for which no sequence of commands exists. Partial credit for other test cases will be given as described below.

Correctness: 20% of points

The output file for a test case is considered correct if it is correctly formatted, contains no more than 10000 commands, and the sequence of commands causes the robots to exit the mazes, with the last command causing at least one robot to exit its maze.

Minimality: 80% of points

The output file for a test case is considered minimal if it is correct and there is no shorter sequence of commands that is correct. A program whose sequence of commands is not minimal will receive no points for minimality.

Seeing the Boundary

TASK

Farmer Don watches the fence that surrounds his N meter by N meter square, flat field ($2 \le N \le 500{,}000$). One fence corner is at the origin (0, 0) and the opposite corner is at (N, N); the sides of Farmer Don's fence are parallel to the X and Y axes.

Fence posts appear at all four corners and also at every meter along each side of the fence, for a total of $4 \cdot N$ fence posts. The fence posts are vertical and are considered to have no radius. Farmer Don wants to determine how many of his fence posts he can watch when he stands at a given location within his fence.

Farmer Don's field contains R ($1 \le R \le 30{,}000$) huge rocks that obscure his view of some fence posts, as he is not tall enough to look over any of these rocks. The base of each rock is a convex polygon with nonzero area whose vertices are at integer coordinates. The rocks stand completely vertical. Rocks do not overlap, do not touch other rocks, and do not touch Farmer Don or the fence. Farmer Don does not touch the fence, does not stand within a rock, and does not stand on a rock.

Given the size of Farmer Don's fence, the locations and shapes of the rocks within it, and the location where Farmer Don stands, compute the number of fence posts that Farmer Don can see. If a vertex of a rock lines up perfectly with a fence post from Farmer Don's location, he is not able to see that fence post.

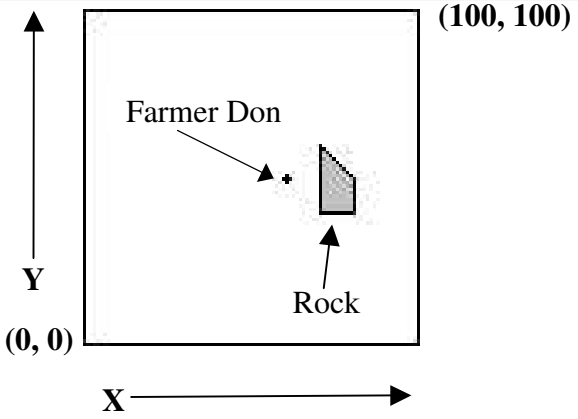
Input: `boundary.in`

- The first line of input contains two space-separated integers: N and R.
- The next line of input contains two space-separated integers that specify the X and Y coordinates of Farmer Don's location inside the fence.
- The rest of the input file describes the R rocks:
  - Rock i's description starts with a line containing a single integer p_i ($3 \le p_i \le 20$), the number of vertices in the rock's base.
  - Each of the next p_i lines contains a space-separated pair of integers that are the X and Y coordinates of a vertex. The vertices of a rock's base are distinct and given in counterclockwise order.

Example input:

```
100 1
60 50
5
70 40
75 40
80 40
80 50
70 60
```

Note that the base of rock 1 has three collinear vertices: (70,40), (75,40), and (80,40)



Output: `boundary.out`

The output file should contain a single line with a single integer, the number of fence posts visible to Farmer Don.

Example output:

```
319
```

CONSTRAINTS

Running time	1 second of CPU
Memory	64 MB

SCORING

You will receive full points on each test case for which your program produces a correct output file. No partial credit will be given on any test case.