Rujia Liu's Present 7
Hello, Interactive Problems!

22nd February, 2014
UVa Online Judge

Problems

A. Mysterious Space Station
B. Guess the Fake Coin
C. Guess the Fake Coin II
D. Guess the String
E. Guess the Missing Number
F. Guess the Convex Polygon
G. Team star, Team moon
H. Explore the Dune
I. Xmas Gift
J. Stone Age
K. Weights of Toys
L. Mining in Starcraft

As usual, there is a gift package on the contest website that contains some additional I/O data, special judge, all the server codes for each interactive problem, and a sample interactive problem 'Guess the Number' with server codes and client codes in C++, Pascal and Java. Please make best use of it :)

Thanks Md. Mahbubul Hasan for all the problems, Yubin Wang for problem C, F and L, Lijie Chen for problem L, Feng Chen for writing and testing Java codes for some of the problems and finally, Miguel Revilla Rodríguez for integrating interactive problems into UVa OJ :)

I hope you enjoy this unique contest, and interactive problems :)

Hello, everyone! My name is Rujia Liu. I used to do a lot of problem solving and problemsetting, but after graduated from Tsinghua University, I'm spending more and more time on my company ☹

(You may realized that the paragraph above is copied from the texts of my 3rd , 4th , 5th and 6th contest, but that's me, lazy me.)

This time, my contest is about interactive problems. It's the first contest that UVa uses interactive problems. Actually UVa decided to add support for interactive problems after my contest proposal. Thanks, UVa team!

I guess interactive problems are new to many of you, so I used some classic problems for you to warmup, before trying the difficult ones. Other interactive problems are authored by me quite a few years ago, for Chinese Olympiad in Informatics, so if you've solved these problems before, please don't submit immediately ☺

I've also included two traditional problems so that if you hate interactive problems, you still have something to do. However, be careful: one of them is very hard, and the other one is even harder...

Ah, forgot to mention, problem B, D and E are good starts :)

Best regards,
Rujia Liu

Interactive Problems: General Instructions for UVa OJ

From contestant's perspective, the difference between interactive problems and traditional (batch) problems is that the input to your program (client) is not statically stored in a file: it is produced dynamically by another program (server), and the output of the client is immediately sent to the server, which can be used by the server to compute its next output (which is in turn sent to the client).

For example, in a "guess the number" problem, you're requested to guess an integer between 1 and 1000 in 20 turns. Each time you can ask an integer, the server will tell you whether it's bigger, smaller or exactly same.

In UVa OJ, *the client should read from standard input, and write to standard output*. The communication between the server and the client is line oriented. That means after printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` or `cout << flush` in C/C++, `flush(output)` in Pascal and `System.out.flush()` in Java. This way, your output will be sent to the server immediately. The server will also flush the output after each line, but you don't have to read a whole line at a time.

In short, the only important thing you should know is: always flush your output after printing a line. Everything else is the same as traditional problems.

Interactive problems are usually harder to solve, so UVa OJ kindly provides some more verdicts in addition to the traditional ones like AC (Accepted), WA(Wrong Answer) and TLE(Time Limit Exceeded):

BC = Broken Communication
PV = Protocol Violation
PLE = Protocol Limit Exceeded

The first one means the server failed to read a line or write a line, which usually means your program exited presumably. The latter two are defined for each problem separately, which is usually quite intuitive.

Note that servers can exit presumably when there is no need to continue.

Interactive Problems: Rujia Liu's Present 7

Here are the instructions for interactive problems that are valid only in this contest. However, I find these instructions very helpful not only to the contestants, but to the problem authors. So if you want to author an interactive problem, please read on careful.

In the gift package, all the source codes of the servers are provided. There was also an example problem: guess the number, mentioned above. *You can find the problem description, server code, correct (AC) client code in C++, Pascal and Java, and some incorrect code (WA, PV and PLE etc).*

Each server can be run in two modes: manual mode and auto mode. In manual mode (which is the default mode), the server reads from the standard input and writes to the standard output. So before coding, you can play with the server manually, which may help you understand the behavior of the server.

However, after you finished your client, it's difficult to test your client with the server. You have to open two windows, one running server and one running client, and manually send (type with keyboard or copy/paste) the output of server to the input of client, and the output of client to the input of server. Isn't it terrible?

So auto mode comes to rescue. In auto mode (which can be enabled by defining AUTO_MODE when compiling the server), the server (now it's called autoserver) launches your client as a child process. With some magic, your client's input automatically comes from the server, and your client's output automatically goes to the server. With the debug information printed by the server ("Correct" or some error message), you can easily judge your client by yourself. **Attention: the debug information printed in your client's standard error will be swallowed. Any debug information from your client should be printed to files.**

However, all the servers in this contest need an input file (see the source code for more details). The format of this input file is not described anywhere, so you have to figure it out by yourself. Then you can test your program with different input files, just like traditional problems.

The interaction protocols in this contest are all text-based. Formally speaking, for each test case, your client should read some initial data (just like traditional problems), then your client should issue some commands, read their results, until the test case is solved.

The commands are very similar to *function calling* in programming languages. The command always starts with a command name, then zero or more arguments (typically in the same line), and the result is always zero or more arguments (usually integers). There should be one single space between tokens (command name and arguments).

We advice you to wrap the communications in functions. Suppose there is a command Query:

Command	Description
`Query i j`	Returns *s*, whether person *i* and person *j* belongs to the same tribe. Both *i* and *j* must have entered the square. (They can be already gone, though).

Then you can wrap it in a function like this:

```
int query(int i, int j) {
  printf("Query %d %d\n", i, j);
  fflush(stdout);
  int x;
  scanf("%d", &x);
  return x;
}
```

This way, solving interactive problems are very similar to solving traditional problems: you read initial data as usual, and call these "wrapped" functions, until you're done.

Sometimes the server will exit before you expect it to do so. That's usually after your client sent something wrong (wrong command format or incorrect answer). We strongly encourage you to check the return values of your I/O functions and exit immediately when those functions failed to read/write data, otherwise your program may face some junk data (due to bad read) and receive Runtime Error instead.

Note that debugging your client is a bit harder than before, because you can't simply make a file and use redirection to feed it to your client. One way is to re-use the *interactlib* that the server codes depend on, then your client can launch the server (in default mode) and your cin/cout automatically connects to the server's output and input, respectively. However, *interactlib* has a serious restriction: you can only use C++ and use cin/cout exclusively for I/O. You can't use other languages and you can't use stdio stuffs (scanf/printf etc).

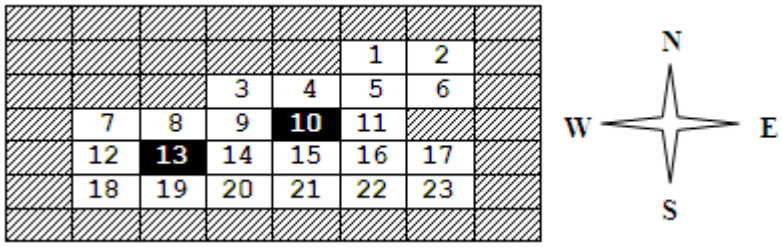
A. Mysterious Space Station

Year 3000. Scientists found a mysterious space station. After months of work, they built a digital map of the station and sent a robot there for further investigation.

The map is an *n***m* grid, each cell is either empty or an obstacle. The robot can move east, south, west or north one cell at a time. There is no light or other sensible substance in the space station, so the robot had to "blind walk". Only if he fails to walk, it knows that there is an obstacle in front.

The map is special: all the obstacles are connected (via north, east, west, south directions), and all the empty cells are enclosed by the obstacles. Each two empty cells can be reached from each other, and there is no "tunnels" in the space station (i.e. for each empty cell, at least one of its north neighbor and south neighbor is empty, and at least one of its east neighbor and west neighbor is empty).

We number all empty cells 1, 2, 3, ... from north to south, west to east like this:



There are also *k* teleport controllers, each controller is connected with a pair of different empty cells (called teleport cell). Each teleport cell can be connected to at most one controller, and the 8 neighbors of each teleport cell will not be another teleport cell or an obstacle.

If two teleport cells are connected to the same controller, once the robot walks into one of them, it'll be teleported to the other cell, then move into the neighbor cell in the same direction. During the process, the robot is not aware of being teleported.

The scientists have already sent a robot to a particular cell. Due to technology restriction, the starting cell is always an empty cell that has a neighboring obstacle. For example, the robot can be sent to cell 12 in figure 1.

If there is no teleport controller, when the robot moves EN, it'll reach cell 8. When he tries to move N, it'll be blocked by an obstacle. However, if a teleport controller connects cell 10 and 13, the robot can successfully execute commands ENN. The actual route is 12->(13->10->)11->5->1.

Your task is to control the robot to discover all the teleport controllers within a reasonable amount of commands.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

There will be at most 20 test cases. For each test case, first read three integers *n*, *m* and *k* (6<=*n*, *m*<=15, 1<=*k*<=5). Then read *n* lines, each contains *m* characters. '.' denotes empty cells, '*' denotes

obstacles, and 'S' denotes the starting cell. Then issue one or more `MoveRobot` command and read the result. When you're ready, issue exactly *k* `Answer` commands.

command	Description
`MoveRobot D`	Try to move to direction D. return 1 if successful, 0 otherwise.
`Answer pos1 pos2`	You found a teleport controller connecting empty cells pos1 and pos2. This command does not return anything.

Note that each teleport controller should be mentioned exactly once, but can be in any order.

The interaction ends when n=m=k=0.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 32767 `MoveRobot` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

`7 8 1`
`********`
`*****..*`
`***....*`
`*.....**`
`*S.....*`
`*......*`
`********`	

	`MoveRobot E`
`1`	
	`MoveRobot N`
`1`	
	`MoveRobot N`
`1`	
	`MoveRobot W`
`0`	
	`MoveRobot N`
`0`	
	`MoveRobot E`
`1`	
	`MoveRobot E`
`0`	
	`Answer 10 13`
`0 0 0`	

Problemsetter: Rujia Liu
Source: Winter Camp of Chinese Olympiad in Informatics 2002 (This is the first time I make problems for a serious contest)
Special Thanks: Renshen Wang (Original Alternative solution writer), Md. Mahbubul Hasan

B. Guess the Fake Coin

There are *n* coins. One of the coins is fake: it is slightly heavier than the other coins, and all other coins have the same weight.

Your task is to find out the fake coin.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T($1<=T<=100$). For each test case, read an integer n ($3<=n<=120$) in the first line, then issue one or more `Test` command, followed by an `Answer` command.

Command	Description
`Test` L_1 L_2 ... R_1 R_2 ..	Places coins L_1, L_2 ... on the left side and R_1, R_2... on the right side ($1<=L_i,R_i<=n$), and returns the result a. a=1 means left>right, a=-1 means left<right, a=0 means left=right. There should be an even number of coins, and the first half are placed on the left side. No coin should appear in a command twice.
`Answer b`	Tell us your answer. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 5 `Test` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 9 -1 0	 Test 1 2 3 4 5 6 Test 4 5 Answer 6

Classic Problem.
Adapted by Rujia Liu
Special thanks: Md. Mahbubul Hasan

C. Guess the Fake Coin II

There are *n* coins. One of the coins is fake: it is slightly heavier than or lighter than the other coins, and all other coins have the same weight.

Your task is to find out the fake coin.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T($1<=T<=15000$). For each test case, read an integer n ($3<=n<=120$) in the first line, then issue one or more `Test` command, followed by an `Answer` command.

Command	Description
`Test` L_1 L_2 ... R_1 R_2 ..	Places coins L_1, L_2 ... on the left side and R_1, R_2... on the right side ($1<=L_i,R_i<=n$), and returns the result a. a=1 means left>right, a=-1 means left<right, a=0 means left=right. There should be an even number of coins, and the first half are placed on the left side. No coin should appear in a command twice.
`Answer b r`	Tell us your answer. r=1 means coin b is heavier than other coins, r=-1 means coin b is lighter than other coins. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 5 `Test` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 9 -1 0 0 1	 Test 1 2 3 4 5 6 Test 4 5 Test 1 6 Test 3 2 Answer 2 -1

Classic Problem.
Adapted by Rujia Liu
Special thanks: Md. Mahbubul Hasan, Yubin Wang

D. Guess the String

I have a 01 string S whose length is between 1 and 100 (inclusive). Your task is to guess it.

Each time you can give me a 01 string and I'll tell you whether it's a continuous substring of S.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T(1<=T<=100). For each test case, issue one or more `Ask` command, followed by an `Answer` command.

Command	Description
`Ask t`	Returns 1 if t is a substring of S, otherwise returns 0. The length of string t should be between 1 and 100 (inclusive).
`Answer S`	Tell us your answer. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 222 `Ask` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1	
	Ask 010
1	
	Ask 101
1	
	Ask 100
0	
	Answer 0101

Classic Problem.
Adapted by Rujia Liu
Special thanks: Md. Mahbubul Hasan

E. Guess the missing number

There are n-1 integers a_1, a_2, a_3, ..., a_{n-1}, $1<=a_i<=n$. No two integers are the same, so exactly one integer from {1, 2, 3, ..., n} is missing.

Your task is to find out which one is missing.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T ($1<=T<=25$). For each test case, first read an integer n ($2<=n<=1000$). Then issue one or more `Ask` commands followed by one `Answer` command.

Command	Description
`Ask i j`	Returns the j-th bit of a_i ($1<=i<=n-1$, $0<=j<=10$). The bits are numbered 0, 1, ... from right to left.
`Answer x`	Tells us that x is the missing integer. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 2222 Ask commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 4	
	Ask 1 0
0	
	Ask 1 1
1	
	Ask 2 2
1	
	Ask 3 0
1	
	Ask 3 1
0	
	Answer 3

Sample Explanation

There is only one test case, $a_1=2$, $a_2=4$, $a_3=1$.

Classic Problem.
Adapted by Rujia Liu
Special thanks: Md. Mahbubul Hasan

F. Guess the Convex Polygon

There is a convex polygon P on the Cartesian plane satisfying the following conditions:

1. The number of vertices *n* satisfies 3<=n<=20, and each vertex (x,y) satisfies |x|,|y|<=10000.
2. (0,0) is strictly inside P.
3. No two edges are collinear.
4. No edges are parallel to x or y axis.
5. Vertices have integer coordinates.

Your task is to "guess" the polygon.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T (1<=T<=100). For each test case, you can issue one or more `AskX` and `AskY` commands followed by one `Answer` command.

Command	Description
`AskX x0`	Returns c, the number of intersection points between P and line x=x0, and their y coordinates, y_1 y_2 ... y_c.
`AskX y0`	Returns c, the number of intersection points between P and line y=y0, and their x coordinates, x_1 x_2 ... x_c.
`Answer n` x_1 y_1 x_2 y_2 ... x_n y_n	Tell us your answer. The vertices must be in counter-clockwise but you can start from any vertex. This command does not return anything.

Each returned coordinate is given in "reduced fraction form" by two integer a and b, that means the coordinate is a/b.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 500 Ask (AskX or AskY) commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

`1`	
`1 2 1`	`AskX -6`
`2 -5 1 17 5`	`AskX -5`
`2 16 1 -6 1`	`AskY 2`
`0`	`AskY -20`

	Answer 5 8 -9 16 2 -1 9 -6 2 -5 -5

Note that this interaction is only *valid* and does not mean the user program can really deduce the answer from the AskX/AskY commands before it.

Problemsetter: Rujia Liu (Idea), Yiming Li (Contest Materials)
Source: Chinese Olympiad in Informatics 2003
Special thanks: Yubin Wang, Md. Mahbubul Hasan

G. Team star, Team moon

You see a group of children playing a seeming interesting game.
"Hi kids, what are you playing?" You asked.
"We have a judge. Other kids belong to team Star or team Moon. There are *n* kids in team Star and *m* kids in team Moon. Guess who is the judge then we'll tell you what game we're playing."
"Ok! Any tips?"
"You can ask questions about whether someone belongs to some team. You can't ask whether someone is judge. Team Star will always tell the truth, while team Moon will always lie, but the judge is naughty: he will tell the truth for the 1st, 3rd, 5th... time you ask him, and lie for the 2nd, 4th, 6th... time."
"Restrictions?"
"You cannot ask a person about himself. For example, you can't ask me: 'Are you Star?' And you can ask a person at most two questions, and these two questions can't be about the same person. For example, if you asked me whether Tom belongs to team Star, you cannot ask me whether Tom belongs to team Moon, because you're asking me about Tom twice."
"Ok! Then I'll tell you every one's role, not only who is judge."

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T (1<=T<=100). For each test case, read two integers *n* and *m* in the first line (2<=*n*+*m*<=500).

Then issue one or more `Ask` command, followed by an `Answer` command.

Command	Description
`Ask C1 C2 T`	Ask child C1: Does C2 belong to team T? 1<=C1,C2<=n+m+1, and C1 should not be equal to C2. T=0 means Team Star, T=1 means Team Moon. Returns 1 if the answer is "Yes", 0 otherwise. You cannot ask a child about the same child twice.
`Answer` R_1 R_2 `...` R_{n+m+1}	Tell the children each child's role. 0 means Team Star, 1 means Team Moon, 2 means judge. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can ask a person at most two questions, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 1 1 0	 Ask 1 2 0 Ask 2 1 0

1 0	 Ask 3 1 1 Answer 2 1 0

Problemsetter: Rujia Liu
Source: Chinese Olympiad in Informatics 2002
Special thanks: Yiming Li, Ziqing Mao (both are original alternative solution writers), Md. Mahbubul Hasan

H. Explore the dune

There is a maze in a massive dune. In the maze, there are *n* (2<=*n*<=25) caves and *m* (1<=m<=300) bidirectional corridors connecting them. You can drop a robot into the maze (through a small hole from above), move it around and collect information about the maze.

When the robot is inside a cave, you can see how many corridors are connecting it. But it's too dark to distinguish two caves with the same number of corridors connecting it. Further more, the corridors are too dark, so the robot cannot know which corridors it is walking along.

The only thing that can help you is a token. Initially, the token is in the robot's pocket. You can ask the robot the leave the token in a cave, or to pick it up with it. Note that you cannot leave the token in a corridor because corridors are too dark.

Your task is the find out the number of caves and corridors. Note that you have no way to take out the robot from the cave. Don't worry, it's just a robot :)

It is guaranteed that the maze is connected, no two corridors are connecting the same pair of caves, and no corridor is connecting a cave and itself. Note that the corridors can be twisted in 3D, so the maze is not necessarily a planar graph.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T (1<=T<=25). For each test case, issue one or more Look, Walk, Put and Take commands, then one Answer command.

Command	Description
`Look`	Returns *d* and *s*, where *d* is the number of corridors connecting the current cave, and *s*=0 means there is no token in the cave, *s*=1 means there is a token.
`Walk i`	Walk along the corridors numbered *i*. Initially the corridors connecting the starting cave are counter-clockwise numbered 0, 1, 2, ..., starting from an arbitrary corridor. Later, the corridors connecting the current cave are always counter-clockwise numbered, starting from the corridor that you used to enter current cave. That means, "Walk 0" always means "go back". This command does not return anything.
`Put`	Put the token in current cave. Please make sure the token is in robot's pocket. This command does not return anything.
`Take`	Take the token from the current cave. Please make sure the token is on the ground of the current cave. This command does not return anything.
`Answer n m`	You found out the maze contains *n* caves and *m* corridors. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

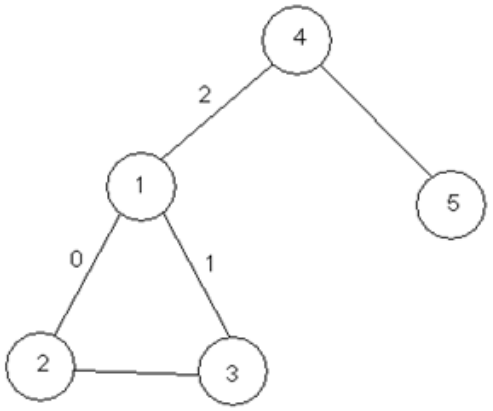
For each test case, you can issue at most 65536 `Walk` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 3 0 2 0 2 0 3 1 2 0 1 0	 Look Put Walk 0 Look Walk 1 Look Walk 1 Look Take Walk 1 Look Walk 1 Look Answer 5 5

Sample Explanation

The maze graph is:

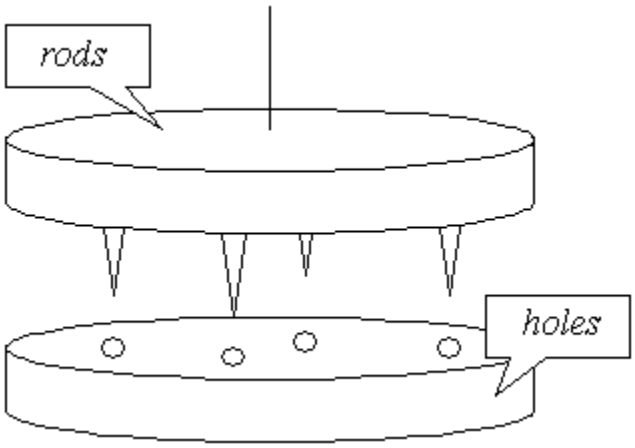


Initially, you're at node 1, corridor 0, 1, 2 lead to cave 2, 3, 4, respectively.

Problemsetter: Rujia Liu (Idea), Xiaohui Bei (Contest Materials)
Source: Chinese Olympiad in Informatics 2004
Special thanks: Original alternative solution writers, Md. Mahbubul Hasan

I. Xmas gift

Christmas is coming. In the center of the city park, there is a huge toy, shown in the picture.



There are *n* rods in the upper part and *n* holes in the lower parts. Both rods and holes are equally spaced. The lengths of the rods are exactly 1~*n* (but not necessarily in this order), and depths of the holes are also exactly 1~*n*.

Initially, the two parts are touching each other. That means, each rod is in a hole with the same depth as the rod's length. Then, before the game starts, the upper part is lifted and rotated quickly for several seconds.

Your task is the re-connect the two parts like its initial state (i.e. each rod is in a hole with the same depth as the rod's length). Note that you can see the lengths of the rods, but you can't see the depths of the holes.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T (1<=T<=25). For each test case, first read an integers *n* (1<=*n*<=100,000) in the first line. The next line contains *n* integers, the lengths of the rods.

Then issue one or more `Rotate` or `Drop` command.

Command	Description
`Rotate k`	Rotate the upper part `k` unit (1<=*k*<*n*) counter-clockwise. This command does not return anything.
`Drop`	Try to connect the two parts. Returns the distance of the two parts. i.e. maximum l_i-d_i, where l_i is the i-th rod's length (after rotation), d_i is the depth of the i-th hole. When this command returns 0, the current test case ends.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 5 `Drop` commands (including the last one that successfully connects the two parts), **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

1 5 2 5 1 3 4 3 0	 Rotate 4 Drop Rotate 3 Drop

Sample Explanation

The holes' lengths are 1, 3, 4, 2, 5. After "Rotate 4", the rods' lengths are 4, 2, 5, 1, 3.

Problemsetter: Rujia Liu
Source: Winter Camp of Chinese Olympiad in Informatics 2003
Special Thanks: Original Alternative solution writers, Md. Mahbubul Hasan

J. Stone Age

There is a very large square in Stone Age. People are entering and leaving the square very frequently. When there is a tribe whose number of people in the square is strictly larger than the number of other people in the square, we say that tribe is dangerous.

Your task is to keep checking whether there is a dangerous tribe.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

There is only one test case. Keep using `Getjob` command to read an integer *v* per line, until *v*=-2 (see below). There will be no more than 50000 jobs.

v	Meaning
0	New event: There is a person entered the square. People are numbered 1, 2, ... in the same order as they enter the square.
>0	New event: The person numbered v left the square.
-1	You need to check whether there is a dangerous tribe and tell us the result with an Answer command.
-2	The test case is finished.

You should issue an `Answer` command exactly once for each v=-1, and after each new event (v>=0), you can issue `Query` commands to gather information. You should not issue any `Query` command after *v*=-1 (you need to `Answer` immediately).

Command	Description
`Getjob`	Returns the next *v*. See the table above. When *v*=-2, the test case is finished.
`Query i j`	Returns *s*, whether person *i* and person *j* belongs to the same tribe. Both *i* and *j* must have entered the square. (They can be already gone, though).
`Answer i`	Tells us whether there is a dangerous tribe. *i*=0 means there is no dangerous tribe, otherwise person *i* must be still in the square, and his tribe is dangerous. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

After each new event, you can issue at most 5 `Query` commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

0 0 0	Getjob Getjob Query 1 2

-1 0 1 -1 2 -1 1 -2	Getjob Answer 0 Getjob Query 2 3 Getjob Answer 2 Getjob Getjob Answer 0 Getjob Getjob

Sample Explanation

In the example above, their tribe ID for each person is: 1, 2, 2.

Classic Problem.
Adapted by Rujia Liu
Source: IOI China Team Selection Contest 2004
Special Thanks: Original Alternative solution writers

K. Weights of Toys

Bingbing has 3 toys: pikachu, wukong and barbie. She didn't know their exact weight, but she knows the interval (in a mysterious weight unit), shown below.

	Pikachu	Wukong	Barbie
Minimum possible weight	1	2	3
Maximum possible weight	3	4	5

Jiajia has an e-mobile, which can tell you the difference between the weights of both sides. The e-mobile is huge, so you can put as many toys as possible on both sides.

Bingbing asks Jiajia for the mobile, but Jiajia wants to give Bingbing a challenge: she can use each toy at most once on each side (so each toy can be used at most twice in total). Bingbing agreed. She used the mobile twice, as shown below (number D means the left part is D unit heavier than the right part):

Then, Bingbing knows the exact weights of each toy, as shown below:

	Pikachu	Wukong	Barbie
Minimum possible weight	3	4	3
Maximum possible weight	3	4	3

One month later, Bingbing got *n* new toys and used the mobile *m* times (obeying the rules above). Could you tell me the tightest bounds of each toy's weight?

Input

There will be at most 20 test cases. Each test case begins with a line containing two integers n and m (3<=n<=200, 1<=m<=100). The second line contains 2n integers, the 2i-1'th and the 2i-th integer are the initial lower bound and upper bound of the i-th toy(1<=b<=c<=20000). There are m lines followed, each for a use of mobile. Each of these lines begins with 3 integers L, R, D (L,R>=0), that means there are L toys on the left, and R toys on the right. The total weight of the left side is D unit larger than the right side (D<0 means the weight of the left side is -D unit smaller than the right side). The next L integers are the toys on the left side, and the next R integers are the toys on the right side. It is guaranteed that each toy can appear on each side at most once. The input terminates with n=m=0.

Output

For each test case, print 2n integers, in the same format as the input. If there is no solution, print a single -1.

Sample Input

```
3 2
1 3 2 4 3 5
1 1 -1 1 2
1 1 1 2 3
2 2
1 5 1 5
1 1 0 1 2
1 1 1 2 1
3 1
1 5 2 5 1 3
2 1 1 1 2 3
0 0
```

Output for Sample Input

```
Case 1: 3 3 4 4 3 3
Case 2: -1
Case 3: 1 2 2 3 2 3
```

Bonus

Be sure to test your program with the data provided in our gift package.

Problemsetter: Rujia Liu
Source: IOI China Team Selection Contest 2005
Special thanks: Tiancheng Lou (Original Alternative Solution Writer), Md. Mahbubul Hasan

L. Mining in Starcraft

In Star Craft, there are two main resources: minerals and vespene gas ("gas" for short).

You can ask SCVs to mine minerals and collect gas.

- If a "mineral" command is given to an SCV, it can get 8 units of minerals after t1 unit time.
- If a "gas" command is given to an SCV, it can get 8 units of gas after t2 unit time.

You can only give a command to an SCV after its previous command is finished.

You can build new SCVs, each costing 50 units of mineral and takes t3 unit time. Those minerals are consumed at the beginning of the process, and you can build at most one SCV at a time (i.e. you cannot start building another SCV before the previous SCV is finished).

Initially, you have 50 units of minerals and 4 SCVs. Your task is to have at least p1 units of minerals and at least p2 units of gas, as soon as possible.

Input

Each test case contains 5 integers t1, t2, t3, p1, p2 (1<=t1,t2,t3<=10, 0<=p1,p2<=100). The last case is followed by t1=t2=t3=p1=p2=0, which should not be processed. There will be at most 1000 test cases, all are randomly generated.

Output

For each test case, print the shortest time T in the first line, followed by the plan (if multiple optimal plans exist, any will do). Each line in the plan has 3 kinds of format:

- `t 0` - build a new SCV at time t
- `t i 1` - give "mineral" command to the i-th SCV.
- `t i 2` - give "gas" command to the i-th SCV.

Print a blank line after each test case, including the last test case.

SCVs are numbered 1, 2, 3... Initial SCVs are numbered 1~4, newly built SCVs are numbered in the same order they're built.

Important: At time T, all the SCVs must be idle, and there should be no SCV being built.

Sample Input

```
10 9 8 0 10
4 10 9 32 72
0 0 0 0 0
```

Output for Sample Input

```
Case 1: 9
0 1 2
0 2 2

Case 2: 24
0 0
0 1 1
0 2 1
0 3 1
0 4 1
4 1 2
4 2 2
4 3 2
```

	4 4 2 14 1 2 14 2 2 14 3 2 14 4 2 14 5 2

Bonus

Be sure to test your program with the data provided in our gift package.

Original Problemsetter: Wenjie Qian
Original Problem Source: IOI China Team Selection Contest 2000
Adapted by Rujia Liu (to show his respect to Wenjie Qian)
Special thanks: Lijie Chen, Yubin Wang, Md. Mahbubul Hasan