Libro de Problemas

Organizado por la Universidad Complutense de Madrid y patrocinado por





A
Caper pizza
caperpizza.{c,cpp,java}

Brunno Doiuna is very fond of caper pizzas, which he always likes to share with his girlfriend. As she also loves capers, it is of the utmost importance, in order to avoid unnecessary quarrels, to split the pizza into two equally-sized slices in such a way that each half contains exactly the same number of capers. However, most caper pizzas also contain a number of peppercorns, and neither Brunno nor his girlfriend likes them. Therefore, it is also crucial that each of the two halves contain the same number of peppercorns. As you can easily observe, depending on the position of capers and peppercorns on the pizza, it is not always possible to make a straight-line cut into the pizza in such a way that the two slices have the same area and the same number of capers and peppercorns lie in each resulting piece. Your task is to design a program to decide if it is possible to make such a cut or not, knowing the positions of capers and peppercorns.

Input Description

We will assume that the pizza is circular and is centered at the origin, and is big enough to contain all capers and peppercorns. We also assume that there is an even number of capers and an even number of peppercorns, and that no cut goes through any of the capers or pepper balls. Additionally, no pair of peppercorns, capers, or a peppercorn and a caper are aligned with the origin, or form an angle of less than 10^{-6} degrees with the origin.

There can be multiple test cases, one after the other. The first line of a test case contains two even integers $c \geq 0$ and $p \geq 0$ (where $2 \leq c + p \leq 30000$) separated by a space, the number of capers and peppercorns, respectively. Each of the following c lines describes the position of a caper using two floating point numbers, separated by a space, representing its x and y coordinates. Each of the next p lines holds two floating point numbers, the x and y coordinates of a peppercorn. A blank line follows each test case.

The last line of input will contain $-1 - 1$. This marks the end of input – do not write any output for this special last line.

Output Description

YES for a positive answer, NO otherwise.

Sample Input

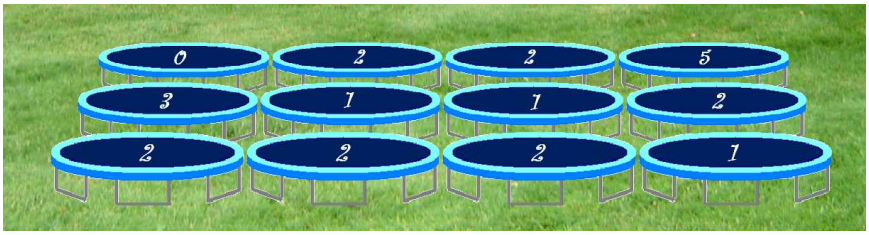
```
2 2
1 1
1 0
0 1
-1 1

2 2
1 1
-1 1
0 1
0.1 -1

-1 -1
```

Sample Output

```
NO
YES
```

B
Grasshopper
`grasshopper.{c,cpp,java}`

We are at a funfair, where an array of trampolines, named "The Grasshopper Labyrinth", catches our attention. As the figure below shows, all of them are labelled with non-negative integers:



People are inside, jumping from one trampoline to another, trying to reach the trampoline in the northwest corner, where the exit to the attraction is located. If you reach the exit fast enough, you may win a prize. However, in order to be eligible for the prize, you must abide by the following rule: after leaping from a trampoline labelled with z, you need to get to another one z trampolines away, in the same row or column.

Therefore, your problem is to find the shortest path from any trampoline to the way out, measured by the number of leaps needed. Since the length of the jump from any trampoline is given, it is sufficient to label each trampoline with the direction of the best jump from it.

$$\begin{array}{cccc} * & \Downarrow & \Leftarrow & \infty \\ \Rightarrow & \Rightarrow & \Uparrow & \Leftarrow \\ \Uparrow & \Rightarrow & \Uparrow & \Leftarrow \end{array}$$

For a given starting position, a path is considered shorter than another if it requires a smaller number of jumps; in case of a tie, the path whose first step gets you northmost in the array is to be preferred; and in case of a tie, the one getting you westmost.

Instead of these symbols, we are using the plain text ones: the appropriate cardinal point ('N', 'S', 'E' or 'W') for the arrows, 'X' for the trampolines without possible escape, and the asterisk '*' for the special trampoline at the upper left position, which represents the exit.

Input Description

Several cases are given in a single test file. The first line in each test case contains a pair of integers between 1 and 50, separated by a single space; the first is the number of rows and the second the number of columns in the matrix. Then, the entries in the matrix follow, line by line, each element being a non-negative integer, again separated by single whitespace characters. A 0×0 matrix will denote the end of the test cases, and hence should not be analyzed.

Output Description

The expected output of each data case is a character matrix. Each element is one of the allowed charset, "`N`", "`S`", "`E`", "`W`", "`X`" or "`*`", as described above. The output for each case is followed by a blank line.

Sample Input

```
3 4
0 2 2 5
3 1 1 2
2 2 2 1
1 20
0 3 3 3 2 5 10 5 3 4 5 4 4 4 6 4 10 3 5 1
5 6
0 7 2 4 4 4
3 1 3 2 3 1
2 1 4 3 4 3
4 4 2 3 3 3
5 4 4 3 4 5
0 0
```

Sample Output

```
*SWX
EENW
NENW

*EEWWWXWWWWWWWWXWWW

*XWSWX
ESSWSW
NWXWWX
EEEWNW
XXNNNX
```

C
Party Night
party.{c,cpp,java}

Today is the town's celebration day, on which tradition dictates that all townspeople go partying. Each of them should attend a party at one of the pubs, and dance and drink to the point of intoxication. Later on, once all the parties have come to an end, after-parties start being thrown at other pubs, and every villager then goes to one. In order for the villagers to make as many acquaintances as possible, no two of them attend the same two parties.

Needless to say, such practice causes everyone to have a severe blackout regarding the events of the night, but people are still curious to know what happened. Unfortunately, all they seem to be able to remember is who coincided with them at some point, but they have serious trouble identifying when or where. And as their memory of even this piece of information may be shaky (to say the least), they need help in figuring out whether all their recollections are consistent or if, on the contrary, some of the townspeople must have made a mistake (either by failing to remember someone else who was there, or by incorrectly thinking they met someone they didn't). Can you help them?

For example, in a town of 4 people, if we are told that villagers 0, 1 and 2 all met one another, and villagers 2 and 3 met as well, the data is consistent because there might have been parties $P0$ and $P1$, and after-parties $A0$, $A1$ and $A2$, such that person 0 went to $P0$ and $A0$, person 1 to $P0$ and $A1$, person 2 to $P0$ and $A2$, and person 3 to $P1$ and $A2$; this arrangement satisfies all required conditions. However, if persons 0 and 3 claimed to have met too, the data would become inconsistent.

Input Description

The input file will contain several test cases. Each of them begins with a line containing two integers: $1 \leq n \leq 100$, the number of villagers; and $0 \leq m \leq 1000$. m lines follow, each containing a pair of integers i and j, $0 \leq i, j < n$, $i \neq j$, meaning that persons numbered i and j remember having been together in a pub. No pair of people will appear twice.

Different test cases will be separated by a blank line. A line with $n = m = 0$ signals the end of the input.

Output Description

For each test case, print "YES" if there is a configuration of parties, after-parties, and villagers attending them under the conditions described, such that the pairs of people who met each other are exactly those in the input data. Print "NO" otherwise.

Sample Input

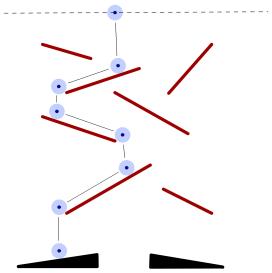
```
4 4
0 1
0 2
1 2
2 3

4 5
0 1
0 2
1 2
2 3
0 3

7 11
0 1
0 2
0 4
1 3
1 5
1 6
2 4
2 5
3 5
3 6
5 6

0
```

Sample Output

```
YES
NO
YES
```

D
Pachinko
`pachinko.{c,cpp,java}`

The game of Pachinko has been all the rage in Japan for nearly a century. The game is played by shooting a metal ball into a special gaming device, depicted below. The ball then falls from somewhere at the top of the board, bouncing against a series of obstacles on its way down. If the player is lucky, the ball goes into one of the pockets, and additional balls are released as a jackpot. Otherwise, the ball is lost (and the game as well).



In the Pachinko parlour *I Control the Physics Core*, the machine boards are rectangular, and the player can drop the ball from any point at a height of 100 centimetres, corresponding to points with cartesian coordinates $(x, 100)$, where $-100 < x < 100$. Obstacles are segments defined by the coordinates of their endpoints. There are neither horizontal nor zero-length obstacles, and they do not intersect one another. For our purposes, we can assume that the ball is a point with no thickness, and we ignore issues of friction, inertia or ricochets. In particular, we can disregard the horizontal displacement of the ball during falls. There is only one jackpot pocket, located at the bottom of the board, between coordinates $(-10, 0)$ and $(10, 0)$. If the ball falls exactly in one of the endpoints, we consider that it hits the jackpot; and if a ball falls exactly into one of the endpoints of a segment, it rolls on that segment, rather than falling through.

Some of the machines at the ICPC parlour have long been suspected of being rigged, as rumour has it that nobody has ever hit the jackpot. A neutral committee has therefore been appointed to verify or refute this claim. As a member, you have taken part in discussions and game trials, all of which have been inconclusive. Much as you enjoyed playing the game for free, after enduring several of the endless meetings you decide it is about time the matter was settled once and for all. To this end, you have taken on the task of writing a program to determine whether the jackpot is reachable or not, based on the specifications of the Pachinko device.

Input Description

Your program will be tested on one or more machines. The description of each machine starts with an integer n ($0 \leq n \leq 500$), indicating the number of segments in it. Each of the following n lines describes a segment by giving 4 real numbers $x\ y\ x'\ y'$, representing the coordinates (x, y) and (x', y') of each of its endpoints, where $-100 < x < x' < 100$, $0 < y < 100$, $0 < y' < 100$, $y \neq y'$. No two segments intersect. A blank line follows each case. The last line of input contains `-1`.

Output Description

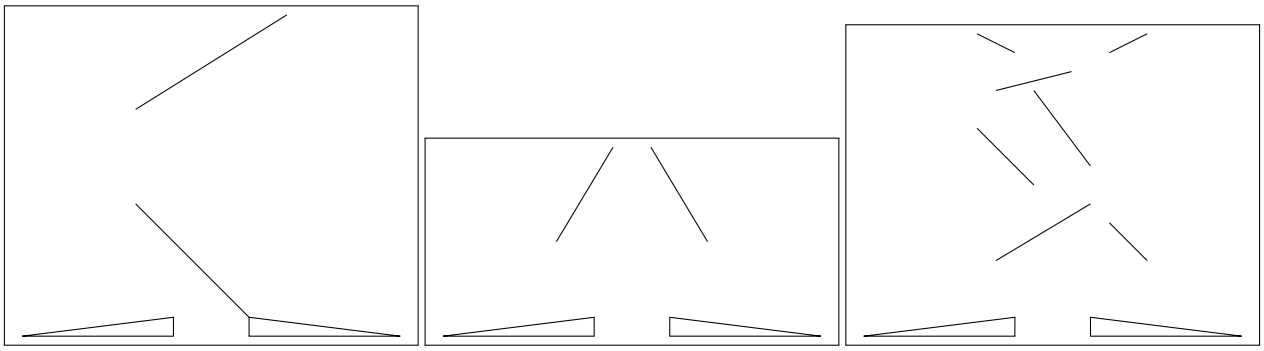
For each machine, answer `yes` if the jackpot is reachable and `no` otherwise.

Sample Input

```
2
-20 60 20 85
-20 35 10 5

2
-5 50 -20 25
5 50 20 25

7
-20 80 -10 75
-15 65 5 70
15 75 25 80
-5 65 10 45
-20 55 -5 40
-15 20 10 35
15 30 25 20

-1
```

Sample Output

```
yes
yes
no
```



E
Lexicographical ranking
`ranking.{c,cpp,java}`

As is well known, an alphabet is a standardized set of letters, and a word is the smallest free form in a language; it can be written as a sequence of letters and symbolizes a meaning. Letters, as elements of alphabets, have a prescribed order, generally known as *alphabetical order*. The principle behind extending the alphabetical order to words (*lexicographical order*) is that all words in a list beginning with the same letter should be grouped together, and before any words starting with a letter that comes later in alphabetical order; within a group of all words starting with the same letter, all words beginning with the same two letters shall be grouped together, and so on; thus, when comparing two words for lexicographical order, the ordering is determined by the alphabetical order of the two letters at the position where the two words first differ. If a word is a prefix of another, the former comes before the latter.

Here we are interested in a generic kind of words, bearing no relation with any specific language. Each of such, let's say, *pseudo-words* will be an ordered subset of letters (that is to say, no letter is repeated and it does not matter whether the word has a meaning in any actual language or not). As the set of available letters we will use the lower case (also called minuscule) form of the Latin alphabet, along with their standard alphabetical order:

`a<b<c<d<e<f<g<h<i<j<k<l<m<n<o<p<q<r<s<t<u<v<w<x<y<z`

Your task is to calculate the position of a given string (its *rank*) in the list of all the pseudo-words we can generate by using only characters of the string (remember that all of them are different), sorted in lexicographical order, as well as to find the pseudo-words corresponding to one or more given ranks.

Input Description

The input consists of several test cases. The first line, for each of them, contains a string of (distinct) characters of the Latin alphabet in lower case ('`a - z`'). The length of such a string will be between 1 and 20, inclusive. The following lines will contain an integer between 1 and the total number of pseudo-words that is possible to form with the letters of the string, until a new string is found or the input file ends.

Output Description

For each test case, output a line with the rank of the input string in the list of all pseudo-words, and then print in a line by itself each of the pseudo-words ranked at the positions of the corresponding input numbers.

Sample Input

```
cadb
1
64
38
13
23
abcdefghijklmnopqrst
1
21
```

Sample Output

```
38
a
dcba
cadb
adb
bc
20
a
abcdefghijklmnopqrt
```

F
Trust groups
`groups.{c,cpp,java}`

The personnel department of *Association of Cookie Monsters* (ACM) has noticed that the productivity of various work groups in the company is not as good as it could be. They have interviewed the employees in the affected groups and they have detected the root of the problem: trust (or, rather, the lack thereof). Some employees do not trust the rest of the group, and this is decreasing their motivation and happiness. The personnel department wants to solve this problem, and has decided to reorganize the groups so that they are *stable*, i.e., they are formed by people who trust each other. They have asked the employees, and they know the people each employee trusts directly. Moreover, if employee A trusts employee B and employee B trusts employee C, then employee A will trust employee C. And obviously, each employee trusts himself. They want to create as few groups as possible to reduce administration overhead (they also do not want to work too hard).

With this information they have contacted you, and asked you to write a program that finds the minimum number of *stable* groups that can be created.

Input Description

The input consists of several test cases. Each test case begins with a line containing two positive integers P and T ($1 \leq P \leq 1000$, $0 \leq T \leq 999000$) separated by a single space. P lines come next, each containing the name of one person. The names will have the following format: surname, a comma, a space and first name (for example `McBride, John` or `Smith, Peter`). Both the surname and the first name will be strings of uppercase or lowercase characters (with no blanks or punctuation marks), with a maximum length of 10 characters. There will not be repetitions in the complete names of the people. After the names there will appear T blocks of 2 lines representing the trust relations between people. Each line of the block will contain the name of a person in the same format as before, and the block will mean that the person in the first line trusts the person in the second line. All people appearing in the confidence relations will have appeared in the previous list of P people.

The input will end with the "phantom" test case `0 0`, which must not be processed.

Output Description

For each test case, the output will be a line containing a positive integer representing the minimum number of *stable* groups of people that can be formed.

Sample Input

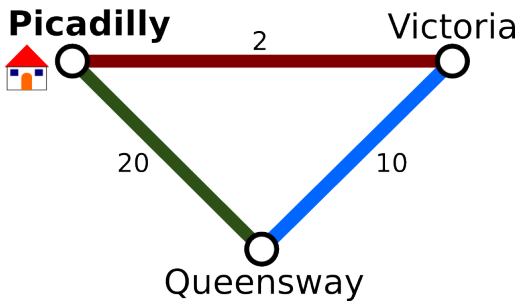
```
3 2
McBride, John
Smith, Peter
Brown, Anna
Brown, Anna
Smith, Peter
Smith, Peter
Brown, Anna
3 2
McBride, John
Smith, Peter
Brown, Anna
Brown, Anna
Smith, Peter
McBride, John
Smith, Peter
0 0
```

Sample Output

```
2
3

```

G
Expensive subway
`subway.{c,cpp,java}`

Peter lives in *Expensive City*, one of the most expensive cities in the world. Peter has not got enough money to buy a car, and the buses in *Expensive City* are pretty bad, so he uses the subway to go to work. Up to now, the subway was very cheap: you could travel anywhere with just one $2 ticket. Last month, the managers decided that it was too cheap so they invented the EFS (Expensive Fare System). With this system, users can only buy monthly tickets between adjacent stations, which allows them to move between these stations any number of times. The price of the monthly ticket varies between stations, so the decision of which tickets to buy must be taken carefully.



With the previous subway plan, the cheapest way to travel from Picadilly to Victoria and Queensway was to buy the monthly ticket Picadilly-Victoria and Queensway-Victoria, for a total cost of $12.

Peter is a salesperson, so he needs to be able to travel to any part of the city. He wants to spend as little money as possible, and here is where you come into the picture. He has hired you to write a program that, given the list of stations, the fares of the monthly tickets between pairs of stations and the station nearest Peter's home, returns the minimum amount of money Peter has to spend in order to travel to any other station. This program also has to return value if it is not possible to go from Peter's home station to all the rest, because in this case Peter will begin to consider using buses...

Input Description

The input consists of several test cases. A test case begins with a line containing two integers: $1 \leq s \leq 400$ (the number of stations) and $0 \leq c \leq 79800$ (the number of connections) separated by a single space. This is followed by s lines, each one containing the name of a subway station. These names will be strings of characters (uppercase or lowercase) without punctuation marks or whitespace characters, and with a maximum length of 10 characters. After the names of the stations there will be c lines showing the connections between stations. A connection allows people to travel from one station to the other in both directions. Each connection is represented as two strings indicating the names of the stations and a positive integer indicating the cost of the monthly ticket, all of which are separated by single spaces. All names of stations appearing in the connections will have previously appeared in the list of s stations. The connections will all be different, and there will not be any connection from a station to itself. The test case will end with a line containing the name of the station from which Peter needs to travel to all the other stations.

The input finishes with the phantom test case `0 0`, which must not be processed.

Output Description

For every test case, the output will be a line containing an integer, the minimum monthly price that Peter has pay to travel from the given station to all the others, or `Impossible` if it is not possible to travel to all the stations.

Sample Input

```
3 3
Picadilly
Victoria
Queensway
Picadilly Victoria 2
Queensway Victoria 10
Queensway Picadilly 20
Picadilly
4 2
Picadilly
Victoria
Queensway
Temple
Picadilly Victoria 2
Temple Queensway 100
Temple
0 0
```

Sample Output

```
12
Impossible
```

H
Turing
turing.{c,cpp,java}

Some interesting documents have recently been found in the ACM archives. These documents contain an account of the original version of the ACM ICPC competition, which took place in Bletchley Park, England, in the year 1943.

In this original contest, programs submitted by contestants were executed in a simplified version of the so-called deterministic Turing Machine, which undoubtedly most of you are familiar with. In this simplified version, the tape is not infinitely long. Instead, tape cells are numbered 0 to $10^3 - 1$ (inclusive), from left to right. Also, the alphabet used is the unary alphabet, meaning that, prior to the execution of the program, the tape will be initialized with the string consisting of n ones followed by all zeroes, where n is the numerical value of the input, and after execution the tape contents should be m ones followed by all zeroes, where m is the numerical value of the corresponding output. In the beginning, the tape head is at position 0, and states are also numbered starting with 0, which is assumed to be the initial state. The machines work as usual: for every machine state q and every bit c (0 or 1), there is at most one rule determining what the next state will be if the symbol at the position of the tape head is c and the current state is q; the rule also specifies the symbol to write at the current position and in which direction the head should move. When no rule applies, the machine stops.

Your task will be to write a program that receives one of the Turing machines submitted by the contestants, as well as the test cases used (input and output pairs), and returns a verdict about the machine's results for each of the cases. Note that in this original contest, unlike the current one, there was a different verdict for each test case, instead of a general one.

Input Description

The input consists of a series of specifications (at most 30) of a Turing Machine and of the corresponding test cases.

The structure of each test case is the following:

The first line contains $1 \leq N \leq 1000$, the number of rules for the machine, and $1 \leq M \leq 100$, the number of test cases for the Turing Machine.

N lines follow, each one containing a rule, in the format q_{prev} c_{prev} q_{next} c_{next} mov. q_{prev} is the state of the machine before the rule is applied, c_{prev} is the content (either 0 or 1) of the tape at the current position p before the rule is applied, q_{next} is the state of the machine after the rule is applied, c_{next} is the content of position p after the rule is applied, and mov is the direction in which the tape head moves one step after applying the rule ("L" if it moves to the left and "R" if it moves to the right). The states should be integers between 0 and 1000, inclusive. The Turing machine should be deterministic; that is, there should be at most one transition from any pair (q_{prev}, c_{prev}), and should not be repeated in your output.

After this, M lines follow, each one containing two space-separated numbers, X and Y, $1 \leq X, Y \leq 1000$. X is the value of the input to the program, and Y is the value of the expected output.

The end of input is signaled by a case with $N = 0$ and $M = 0$.

Output Description

Your output should be *MLE* if the machine attempts to access any cell outside the tape range specified above; *TLE* if it runs for at least 10^4 iterations without stopping or causing an MLE error; *WA* if the

machine stops but returns an incorrect result, and AC if the machine stops and returns the expected result. The output for each case should go in a different line.

Sample Input

```
6 3
0 1 1 0 R
0 0 0 0 R
1 1 1 1 R
1 0 2 1 L
2 1 2 1 L
2 0 900 1 R
1 3
300 301
0 1
0 0
```

Sample Output

```
WA
AC
MLE
```

I
Gnirut
gnirut.{c,cpp,java}

Some interesting documents have recently been found in the ACM archives. These documents contain an account of the original version of the ACM ICPC competition, which took place in Bletchley Park, England, in the year 1943.

In this original contest, programs submitted by contestants were executed in a simplified version of the so-called deterministic Turing Machine, which undoubtedly most of you are familiar with. In this simplified version, the tape is not infinitely long. Instead, tape cells are numbered 0 to $10^3 - 1$ (inclusive), from left to right. Also, the alphabet used is the unary alphabet, meaning that, prior to the execution of the program, the tape will be initialized with the string consisting of n ones followed by all zeroes, where n is the numerical value of the input, and after execution the tape contents should be m ones followed by all zeroes, where m is the numerical value of the corresponding output. In the beginning, the tape head is at position 0, and states are also numbered starting with 0, which is assumed to be the initial state. The machines work as usual: for every machine state q and every bit c (0 or 1), there is a at most one rule determining what the next state will be if the symbol at the position of the tape head is c and the current state is q; the rule also specifies the symbol to write at the current position and in which direction the head should move. When no rule applies, the machine stops.

Unfortunately, only the verdicts of the contest judge were found in the archives, there being no trace of the Turing machines submitted or the test cases used. Note that in this original contest, unlike the current one, there was a different verdict for each test case, instead of a general one.

This discovery has caught the attention of the famous adventurer Zaphod Beeblebrox, who intends to launch a project to construct examples of machines and input/output "files" that might have caused these verdicts. Not only have all your efforts to make him understand the futility of such an enterprise failed miserably, but you also have been coerced into writing a program for him that generates such examples. Your program should receive a series of verdicts about one of the Turing machines submitted by the contestants, and return the specifications of a Turing machine, as well as a series of test cases, for which the verdicts are the ones you received.

We consider the output for a machine should be *MLE* (memory limit exceeded) if the machine attempts to access any cell outside the tape range specified above; *TLE* (time limit exceeded) if it runs for at least 10^4 iterations without stopping or causing a MLE error; *WA* (wrong answer) if the machine stops but returns an incorrect result, and *AC* (accepted) if the machine stops and returns the expected result.

Input Description

The input consists of a certain number (no larger than 30) of series of verdicts. The first line for each contains $1 \leq N \leq 100$, the number of verdicts in the case. The following N lines contain one of the words TLE, MLE, WA and AC.

The end of input is signaled by a case with $N = 0$, which should not be processed.

Output Description

For a given test case for your program, your output should adhere to the following format:

The first line contains $1 \leq M \leq 1000$, the number of rules for the machine, and N, the number of test cases for the Turing Machine.

M lines follow, each one containing a rule, in the format q_{prev} c_{prev} q_{next} c_{next} mov. q_{prev} is the state of the machine before the rule is applied, c_{prev} is the content (either 0 or 1) of the tape at the current position p before the rule is applied, q_{next} is the state of the machine after the rule is applied, c_{next} is the content of position p after the rule is applied, and mov is the direction in which the tape head moves one step after applying the rule ("L" if it moves to the left and "R" if it moves to the right). The states should be integers between 0 and 1000, inclusive. The Turing machine should be deterministic, that is, there should be at most one transition from any pair (q_{prev}, c_{prev}), and should not be repeated in your output. After this, N lines follow, each one containing two space-separated numbers, X and Y, $1 \leq X, Y \leq 1000$. X is the value of the input to the program, and Y is the value of the expected output.

Note: While this is the recommended syntax, other combinations of new lines and whitespace characters might also get accepted.

Sample Input

```
1
AC
0
```

Sample Output

```
2 1
0 1 0 1 R
0 0 2 1 R
4 5
```