TASK	Treasure	Tram	Splot
type	interactive	batch	batch
time limit (per test run)	1 second	1 seconds	2 seconds
memory limit (per test run)	256 MB	256 MB	256 MB
points	100	100	100
	300		

TREASURE

After a recent earthquake, a new island has emerged in the Adriatic sea! The island is mostly barren except for an unusual device that was discovered. The name “oracle” quickly caught on for the device. Although the oracle came with no instruction manual, a crack team of archaeologists and computer experts was able to understand its behavior.

The oracle provides information about the locations of treasure on the island. The island is divided into a grid of cells consisting of *N* rows and *N* columns, with both rows and columns numbered 1 through *N*. Some of the cells in the grid contain treasure. The oracle answers questions of the form “Given a rectangle in the grid, how many cells in the rectangle contain treasure?”

Although the oracle answers questions for rectangles of all sizes, it was found that the more specific the information requested (the smaller the rectangle), the more energy is used by the oracle when answering. More precisely, if a rectangle contains *S* cells, then the oracle uses exactly $1 + N*N - S$ units of energy to answer.

TASK

Write a program that will, given the ability to interact with the oracle, find the **locations of all cells on the island that contain treasure**. We do not want to use too much energy in the process – the less energy is used, the better. It is not required that the amount of energy used is the smallest possible – see the ‘Grading’ section for details on how your solution will be scored.

INTERACTION

This is an interactive task. Your program asks the oracle questions using the standard output, and receives answers by reading from the standard input.

- At the beginning of your program, it should read an integer *N* ($2 \le N \le 100$), the size of the grid.
- To ask the oracle a question, your program should output a line containing four integers, R_1, C_1, R_2 and C_2, separated by spaces such that both $1 \le R_1 \le R_2 \le N$ and $1 \le C_1 \le C_2 \le N$ hold. If the conditions do not hold or the line is incorrectly formatted, your program will receive a score of zero on that test run.
- The oracle will respond with a line containing a single integer – the number of cells in the provided rectangle that contain treasure. More precisely, the number of cells (*R*, *C*) such that $R_1 \le R \le R_2$ and $C_1 \le C \le C_2$ and the cell at row *R*, column *C* contains treasure.
- When your program is done asking questions, it should output ‘END’ on a single line. After that it should output *N* lines, one for each row in the grid, each containing a string of *N* characters ‘0’ (zero) or ‘1’ (one).The *C*-th character in the *R*-th line is ‘1’ if there is treasure in the cell at row *R*, column *C* or ‘0’ if not. Rows are numbered 1 to *N* top to bottom, columns left to right. The execution of your program will be automatically terminated once your program outputs a solution.
- In order to interact properly with the grader, your program needs to flush the standard output after each question and after writing the solution. The provided code samples show how to do this.

You may assume that, in each test run, the oracle will correctly answer questions and the locations of treasure will be chosen prior to the interaction. In other words, the grader is not adaptive and the answers will not depend on previous questions asked by your program.

CODE SAMPLES

Code samples in all three programming languages are available for download on the ‘Tasks’ page of the contest system. The purpose of the samples is only to show how to interact with the oracle; these are not the correct solutions and may not score any points.

GRADING

Each test case is worth 10 points. If the output of your program is incorrect, you get zero points for that test case. Otherwise, the number of points awarded to your program depends on the total number of energy units K used by the oracle. More precisely:

- If $K \leq 7/16\ N^4 + N^2$, your score is 10 points.
- Otherwise, if $K \leq 7/16\ N^4 + 2\ N^3$, your score is 8 points.
- Otherwise, if $K \leq 3/4\ N^4$, your score is 4 points.
- Otherwise, if $K \leq N^4$, your score is 1 point.
- Otherwise, your score is 0 points.

Additionally, in test data worth at least 40% of total points, N will be at most 20.

Your output will be considered correct even if your program guesses the correct solution, e.g. if it does not even ask a single question.

EXAMPLE

In the following example, commands are given in the left column, row by row. Feedback is given in the second column of the corresponding row.

output	input
	2
1 1 1 1	0
1 2 1 2	1
2 1 2 2	2
END	
01	
11	

TESTING

There are two ways to test your program, locally or through the contest system. In both cases, you must first create a file that describes the test case. The first line of the test file should contain the integer N. The following N lines should describe the locations of treasure in the same format as the output of your program. For example, this would be the test file for the example in the previous section:

```
2
01
11
```

To test through the contest system it is first necessary to submit the source code for your program using the "Submit" page, and then use the "Test" page. The contest system will tell you if your program produced the correct output and provide information on the number of queries and energy units used.

To test locally, use the `treasure_test` binary, which can be downloaded through the contest system. The binary is used with a command like `./treasure_test ./my_solution input_file`. The output will contain feedback on whether your program has correctly solved the test case while the `treasure.log` file will contain information about the queries issued by your program and responses received.

TRAM

Seats in a new tram operating in Zagreb are organized into a grid consisting of N rows numbered 1 through N and two columns numbered 1 and 2. The distance between two seats, one at row R_A, column C_A and other at row R_B, column C_B is the Euclidean distance between the centers of the corresponding grid squares – namely $\sqrt{(R_A - R_B)^2 + (C_A - C_B)^2}$.

Most passengers prefer solitude when using public transportation and they always try to choose a seat that is as far away from other passengers as possible. More precisely, when a passenger enters the tram he or she will choose a free seat whose **distance from the closest occupied seat is the highest possible**. If there is more than one such seat, they will always choose one with the lower row number and if there is still more than one such seat, they will choose the one with the lower column number. After the passenger chooses a seat, he or she will sit there until leaving the tram. If the tram is empty, the next passenger to enter will always choose the seat in row 1 and column 1.

TASK

Write a program that will, given a sequence of events, each event either a passenger entering or leaving the tram, determine where each of the passengers was sitting. The tram is initially empty.

There are M events in the input numbered 1 through M in the order in which they occurred. There are two kinds of events: event of type 'E' corresponds to a passenger entering the tram, while the event of type 'L' corresponds to a passenger leaving the tram. For an event of type 'L', an integer P is also given – it specifies that the passenger leaving in this event is the one that entered at event P.

Test data will be such that there will always be at least one free seat in the tram whenever a passenger is trying to enter.

INPUT

The first line of input contains two integers N and M ($1 \le N \le 150\,000$, $1 \le M \le 30\,000$), the number of rows in the tram and the number of events. The following M lines contain the description of the events, K-th of those M lines contains the description of event K – either the character 'E', or the character 'L' followed by a single space and the integer P_K ($1 \le P_K < K$). Each P_K will be valid– event P_K is of type 'E' and no passenger will try to leave twice.

OUTPUT

The number of lines in the output should be equal to the number of events of type 'E' in the input. For each event of type 'E', in the order in which they occurred, output on a single line the row and the column number of the seat chosen by the passenger, separated by a single space.

GRADING

- In test cases worth a total of 25 points, it holds $N \le 150$ and $M \le 150$.
- In test cases worth a total of 45 points, it holds $N \le 1500$ and $M \le 1500$.
- In test cases worth a total of 65 points, it holds $N \le 150\,000$ and $M \le 1500$.

DETAILED FEEDBACK WHEN SUBMITTING

During the contest, you may select up to 50 submissions for this task to be evaluated on a part of the official test data. When the results are ready, a summary of the results will be available on the contest system.

EXAMPLES

input

```
3 7
E
E
E
L 2
E
L 1
E
```

output

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

input

```
13 9
E
E
E
E
E
E
E
E
E
```

output

```
1 1
13 2
7 1
4 2
10 1
2 2
3 1
5 1
6 2
```

input

```
10 9
E
E
E
E
L 3
E
E
L 6
E
```

output

```
1 1
10 2
5 2
7 1
4 2
2 2
4 1
```

SPLOT

The Adriatic coast and islands are lined with amazing beaches of all shapes and sizes. However, many beaches are not accessible by car. To accommodate the growing demand, a giant field near the coast has been converted into a parking lot. Since all of the architects involved have electrical engineering backgrounds, the layout of the parking lot resembles a series-parallel graph often used when designing electrical circuits.

The parking lot consists of parking spaces and two-way roads connecting them. Each road connects two different parking spaces and each pair of parking spaces is connected by at most one road. At any moment, each parking space can be occupied by **at most one car.** No other cars can travel through an occupied parking space.

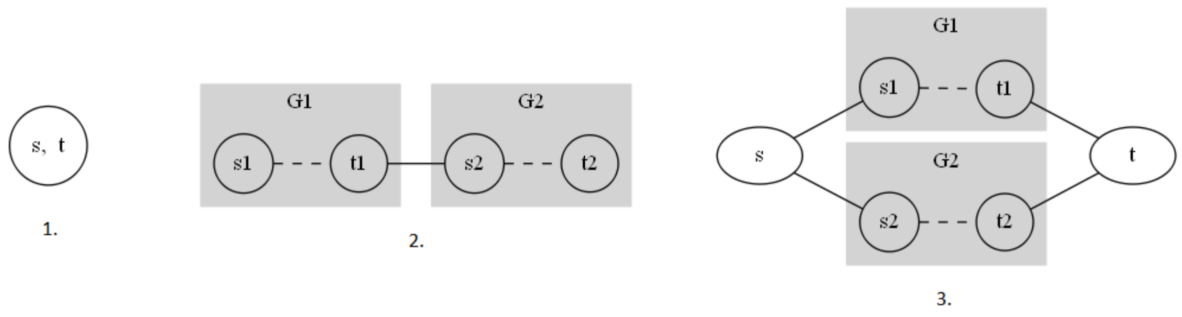

Figure 1: Rules for building splots, each described in the corresponding item below

A series-parallel parking lot (also called *splot*) is a parking lot with two distinguished parking spaces called *source* and *terminal* that is constructed from individual parking spaces using the rules of serial and parallel composition. Each splot can be specified by its encoding – a sequence of characters describing its structure and the locations of parked cars. The valid splots and their encoding are defined recursively as follows:

1. A parking lot consisting of a single parking space and no roads is a valid splot. This single parking space is both the source and the terminal of the splot. The encoding of this splot is simply the lowercase letter '`o`' if the parking space is empty and the lowercase letter '`x`' if the parking space is occupied by a car.
2. If G_1 and G_2 are two valid splots, their *serial composition* G is also a splot. The serial composition is obtained by adding a road between the terminal of G_1 and the source of G_2. The source of the newly obtained splot G is the source of G_1 while the terminal of G is the terminal of G_2. If E_1 and E_2 are encodings of splots G_1 and G_2 respectively then the encoding of G is '`S`E_1E_2`#`'. In other words, the encoding is obtained by concatenating the uppercase letter '`S`', the encodings of splots being composed and the hash character '`#`'.
3. If G_1 and G_2 are two valid splots, their *parallel composition* G is also a valid splot. The parallel composition is obtained by adding two new parking spaces called s and t, adding the roads between s and the sources of both G_1 and G_2, as well as roads between t and the terminals of both G_1 and G_2. The source of the newly obtained splot G is the newly added parking space s while the terminal of G is the newly added parking space t. If E_1 and E_2 are encodings of splots G_1 and G_2 respectively and E_s and E_t are encodings of the source s and terminal t (lowercase letter '`o`' if the corresponding space is empty and lowercase letter '`x`' otherwise) then the encoding of G is '`P`E_s`|`E_1E_2`|`E_t`#`'. In other words, the encoding is obtained by concatenating the uppercase letter '`P`', the encoding of the source parking space, the pipe character '`|`', the encodings of splots being composed, another pipe character '`|`', the encoding of the terminal parking space, and finally the hash character '`#`'.

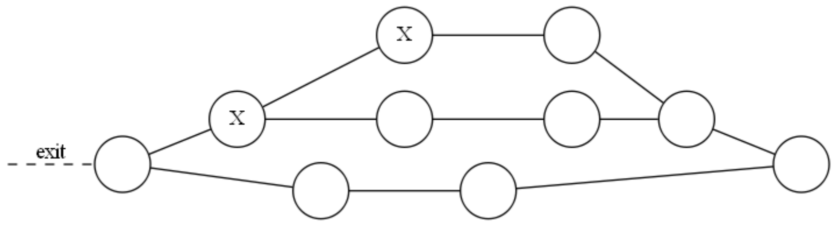

Figure 2: Splot corresponding to the first test example below

For example, the encoding of the splot given in the figure above is '`Po|Px|Sxo#Soo#|o#Soo#|o#`'. Notice that number of lowercase letters in the encoding of a splot *G* is always the same as the number of parking spaces in *G* and that there is a one-to-one correspondence between parking spaces in *G* and the lowercase letters in its encoding.

There is **exactly one exit** from the parking lot and it is directly connected to the source parking space of the overall splot. We say that the car is **not blocked** if it can **exit the splot**, i.e. it can reach the source parking space **via some sequence of roads and empty parking spaces**. For example, in the splot above neither of the cars is blocked, but if we were to park a car in the terminal of the splot (the rightmost node) then one of the cars would become blocked. It is allowed to park a car in the source parking space of the splot, however, if we were to do that, then all other cars in the splot would be blocked.

TASK

The operators of the parking lot would like to arrange incoming cars in such a fashion that **no car is blocked**. Suppose we are given a splot that may already contain some cars but none of those cars are blocked. Write a program that will calculate the **largest total number of cars** that can be parked in the given splot, including the cars already there, without **any of the cars being blocked** and without moving any of the cars already there. Additionally, your program should find one way to arrange this largest number of cars in the splot.

INPUT

The first line of input contains a sequence of at least 1 and at most 100 000 characters – the encoding of the given splot. The only characters appearing in the sequence will be the uppercase letters '`P`' and '`S`', the lowercase letters '`o`' and '`x`', and characters '`#`' (ASCII 35) and '`|`' (ASCII 124). The input will be an encoding of a splot according to the rules above. None of the cars already in the splot will be blocked.

OUTPUT

The output should contain 2 lines. The first line should contain a single integer *M* – the largest number of cars that can be parked in the splot as described above.

The second line should contain a sequence of characters - the encoding of the splot with an optimal arrangement of cars. The sequence should contain exactly *M* occurrences of the letter '`x`' and be obtainable from the input sequence by replacing some of the letters '`o`' by '`x`'.

There may be more than one optimal arrangement and your program may output any one of them.

GRADING

- If the output is incorrect or incomplete, but the first line of output (the maximal number of cars) is correct, the solution will be awarded 80% of the points for the corresponding test case.
- In test cases worth a total of 30 points, it holds that the total number of parking spaces in the splot is at most 20.
- In additional test cases worth a total of 40 points, the splot is empty, i.e. the input does not contain the letter ‘x’.

EXAMPLES

input

```
Po|Px|Sxo#Soo#|o#Soo#|o#
```

output

```
3
Po|Px|Sxo#Sox#|o#Soo#|o#
```

input

```
Po|SPo|oo|o#Px|oo|o##Po|Sxo#Po|ox|o#|o#|o#
```

output

```
7
Po|SPo|xx|o#Px|ox|o##Po|Sxx#Po|ox|o#|o#|o#
```

TOOL SUPPORT

A binary called `splot_tool` (available for download through the contest system) can help you visualize the test inputs and outputs. The tool takes a filename as an argument - either a validly formatted input file or an output file for this tasks and generates an svg image depicting the splot encoded in the file. The generated image can be viewed using a web browser. Sample usage of the tool is:

```
$ ./splot_tool splot.dummy.out.1
splot.dummy.out.1 parsed (10 parking spaces).
splot.dummy.out.1.svg created.
$ chromium splot.dummy.out.1.svg
```

The tool will check that the splot is properly formatted and give an appropriate error message otherwise. It **will not** perform any other validity checks - the files may render properly even if they are incorrect solutions.

The tool will only work for splots whose encoding is at most 200 characters long.