Problem 2. Battle Mage

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Battle mage Va Sya bought a silver plate with form of convex polygon with n vertices to practice his spells.

Then he glued the plate on the thin glass of same form and shape, put this plate as a target and started his practice. Today Va Sya practices a spell which makes a perfect circular hole in the plate itself, but keeps the glass untouched. Note that circles may overlap.

After Va Sya ended up his practice session, he noticed, that all circles from the spells lied inside the plate (but may touch it's edge).

Then he decided to hang the whole construction (glass with remaining parts of silver plate) on the door using selected vertex, so it can rotate freely around this vertex until it is stable.

Find the final coordinates of vertices of the plate, if glass is so thin that its weight must be considered as zero.

Input

First line of the input contains three integers n, c and v ($1 \le n \le 700$, $1 \le c \le 2000$, $1 \le v \le n$) — number of vertices in the polygon, number of spells casted by Va Sya and 1-based number of the vertice he used to hang the remaining plate.

i-th of next n lines contains two integers x_i and y_i — coordinates of the i-th vertice ($-10^4 \le x_i, y_i \le 10^4$). Vertices are listed in counterclockwise order.

Each of next c lines contains three integers cx_i, cy_i and r_i — coordinates of center and radius of one circle. It is guaranteed that all circles are non-degenerated and lie inside the polygon (possibly touches it's edge).

Output

Print n lines, i-th of then containing x and y — final coordinates of i-th vertice with absolute error 10^{-5} or less. Vertices must be listed in same order as in the input file.

Example

standard input	standard output

4 1 3
0 0
3 0
3 3
0 3
1 2 1
	2.253456 -1.176442
	4.714949 0.538507
	3.000000 3.000000
	0.538507 1.285051

Problem 3. Ancient city

Input file:	`intersect-hull.in`
Output file:	`intersect-hull.out`
Time limit:	4 seconds (6 for Java)
Memory limit:	256 mebibytes

Bytelandian archeologists discovered ancient ruins of Twoy. Now they want to know the exact border of this legendary city. For simplicity we can consider the digging site as a two-dimensional plane. If scientists found precious artifacts at points $(x_1, y_1), \ldots, (x_k, y_k)$, they can safely assume that the city was exactly the size of the convex hull of this k points.

In the excavation process there are three types of events:

1. New artifact is located at the point (x, y).
2. Examination of one of the discovered artifacts is shown that it is in fact garbage and it shouldn't be counted as artifact.
3. President of the Byteland wants to build the road, which looks like a straight line $ax + by + c = 0$. He wants to know what will be the damage to the cultural heritage, measured as the length of the intersection between the city and the road.

In the beginning there were no artifacts. You are given the events, you should answer to all the president's questions.

Input

First line contains one integer n $(1 \le n \le 50\,000)$.

Each of the next n lines contains the descriptions of the next events:

1. `+ x y` "— new artifact is found.
2. `- i` "— artifact, found at i-th event is in fact garbage.
3. `?  a b c` "— line equation of the next road $ax + by + c = 0$ $(a^2 + b^2 > 0)$.

All events are numbered from 1 to n. There can be found more than one artifact in one place. Each artifact can be dismissed at most once. All coordinates x, y and a, b, c are integers not exceding 10^8 by absolute value.

Output

For each query `?  a b c` you should output one number "— answer for this query. Absolute or relative error of your answers should not exceed 10^{-9}.

Example

intersect-hull.in	intersect-hull.out

Problem 5. Guess A Number

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds (12 seconds for Java)
Memory limit:	256 mebibytes

This is an interactive problem.

Petya and Vasya are playing a game. The game consists of several rounds. At the beginning of each round Petya secretly chooses a number x from 1 to n (n is independent of the round number). Then Vasya can name a number y from 1 to n and Petya tells if y is less than x. Each round Vasya can name as many numbers as he wants. After that Vasya must name the number x. Vasya's goal is to guess the number correctly while making as few guesses as possible.

Help Vasya write a program that plays the game.

It is guaranteed that Petya won't change the chosen number during the round. However, at the beginning of a new round he can choose the number based on Vasya's behaviour during the past rounds.

Input

The first line contains two integers: maximal number n that can be guessed by Petya and the number of rounds k ($1 \le n \le 10^9$, $k = 10^4$).

After that each query is followed by Petya's answer. Each answer is «`<`» is Petya's number is less than the number in the query, or «`>=`» otherwise.

No response follows an attempt to guess the number.

Output

You should process all k rounds. Each line of output should contain exactly one query of form

`c x`

`c` — a single character describing the query type: `'?'` for a question, and `'='` for a guess attempt.

`x` — an integer from 1 to n.

Total number of `?`-queries should not exceed $k(\log_2(n+1)+0.1)$.

Examples

standard input	standard output
4 2	
	? 3
<	
	? 2
>=	
	= 2
	? 3
>=	
	? 4
>=	
	= 4

Note

Note that the sample test case is invalid since $k \ne 10^4$. Instead, the first test case will be this: `4 10000`

Problem 7. Path Choosing

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	256 mebibytes

You're given a directed graph. Each edge of the graph has its *weight* and *priority*. Let us say that one path is *lexicographically less* than the other path if the sequence of edge priorities for the first path is lexicographically less than the similar sequence for the second path.

You have to write a program that processes queries of two types:

1. Find total weight of edges in the path that is k-th lexicogrphically among all paths starting at vertex v (empty path is included in this count), or determine that such path does not exist.
2. Change the weight of the edge e to w.

Input

The first line of input contains three integers n, m and q — the number of vertices, edges and queries respectively ($1 \le n, q \le 10^5$; $1 \le m \le 10^6$).

Each of the next m lines contains four integers describing respective edge: x_i, y_i, p_i and c_i — start vertex index, target vertex index, priority and weight ($1 \le x_i, y_i \le n$; $1 \le p_i \le m$; $0 \le c_i \le 10^6$). All priorities are distinct.

Note that loops and multiple edges are allowed.

Each of the next q lines describes a query. The first number t_i describes the query type.

If $t_i = 1$ then next follow two integers v_i and k_i — the vertex index and the path index ($1 \le v_i \le n$; $1 \le k_i \le 10^{12}$).

If $t_i = 2$ then next follow two integers e_i and w_i — the edge index and new weight ($1 \le e_i \le m$; $0 \le w_i \le 10^6$).

Output

For each query of type 2 output a single number — answer to the query in a separate line.

Examples

standard input	standard output

3 4 6
1 2 2 1
2 2 4 10
2 3 3 100
1 3 1 1000
1 1 1
1 1 2
1 1 8
2 2 0
1 1 8
1 3 2
	0
	1000
	121
	101
	-1

Problem 11. Distanced Strings

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Vasya wants to construct n strings of equal length that consist of first k Latin letters. He says that *similarity* of two strings is the number of positions where the strings match. Vasya would like similarity of any two strings to be the same, and the similarity should not exceed $2m/(k+2)$, where m is the length of the strings. He also doesn't want the strings to be too long, more precisely, the length of the strings should not exceed $2n$.

Input

The only line of input contains two integers n and k ($2 \le n \le 1000$; $2 \le k \le 26$).

Output

In the first line print two integers m and l — the strings' length and the similarity between any two strings.

In the next n lines print the strings. The strings should consist of lowercase Latin letters.

Examples

standard input	standard output
4 2	3 1 aaa bba abb bab

Problem 13. Edit

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Byteasar keeps his secret, n-letter password for online banking in a text file "`password.txt`". Upon hearing people shouting "Our money is not safe!" outside, he wisely decided to change his password, as it was quite outdated already. Therefore Byteasar created a new password, also consisting of n letters. In order not to forget it he just needs to change the contents of his password file.

The text editor installed on his computer allows Byteasar to conduct two types of operations concerning the file contents:

(1) Changing the i-th ($1 \le i \le n$) letter comprising the file contents.

(2) Replacing each instance of a certain selected letter x present in the file with a letter y ($x, y \in \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$).

The execution of type (1) operation takes Byteasar 1 second. The type (2) operation requires pressing a complicated combination of keys and therefore takes Byteasar c seconds, regardless of the choice of the letters x and y. Operations are executed one by one, and each of them operates on the file contents created as a result of executing all prior operations.

Byteasar wonders how quickly he can perform the edition of the "`password.txt`" file.

Input

The first line of the input contains two integers n and c ($1 \le c \le n \le 1\,000\,000$) denoting both passwords' length and the number of seconds required to perform type (2) operation, respectively. The second and the third line contain strings indicating the previous and the current Byteasar's password, correspondingly. Passwords consist of n lower-case English alphabet characters ('a'-'z') and are not identical.

Output

The only line of the output should contain the minimum number of seconds that Byteasar needs to edit his file using operations (1) or (2).

Examples

standard input	standard output
5 2 aaabc bbbaa	4

Note

In the example, letters at positions 1, 2 and 3 could be changed into the correct ones with a single type (2) operation. Letters occupying positions 4 and 5 are changed using two operations of type (1).

Problem 17. Generator

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	256 mebibytes

You are given n different integer sequences. All sequences have the same length L, and all integers in these sequences are from 1 to m.

You also have a machine that generates a stream of random numbers. Initially, the stream is empty. Every second, the machine generates a random integer from 1 to m, inclusive, and appends it to the stream. Each random integer is generated independently with the same probability distribution which is given to you.

With probability 1, eventually, each of the n given sequences appears as L consecutive elements of the stream. The occurrences of different sequences may overlap. At the first moment when each of the n given sequences appeared at least once, the machine stops immediately. Your task is to calculate the expected number of seconds after which the machine stops.

Input

There are one or more test cases. The first line of input contains an integer T, the number of test cases.

Each test case starts with a line containing three positive integers n, m and L: the number of sequences given, the upper bound for the integers which may appear in the sequences and in the stream, and the length of each given sequence, respectively ($1 \le n \le 15$, $1 \le m \le 100$, $1 \le L \le 5 \cdot 10^4$).

The next line contains m positive integers $a_1, a_2, \ldots, a_m$ which describe the probability distribution the machine uses to generate the stream. The machine will generate 1 with probability a_1/s, 2 with probability a_2/s and so on, where $s = a_1 + a_2 + \ldots + a_m$. It is guaranteed that $s \le 10^9$.

Each of the next n lines contains an integer sequence of length L. All integers in these sequences are positive and do not exceed m. It is guaranteed that all n given sequences are pairwise distinct.

The total sum of $n \cdot L$ over all test cases does not exceed 777 777.

Output

For each test case, calculate the answer as an irreducible fraction $\frac{A}{B}$ and output the integer $(A \cdot B^{-1}) \bmod (10^9 + 7)$ on a separate line. Here, B^{-1} is the multiplicative inverse of B modulo $10^9 + 7$.

It is guaranteed that for all given inputs, the integers B and $10^9 + 7$ are relatively prime.

Example

standard input	standard output

1
1 2 2
1 1
1 1
	6

Problem 19. Honey Tour

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

A little bear Koma likes honey very much. One day, Koma found a storehouse full of honey. The storehouse had M entrances and M exits, and it turned out to be very complicated, like a maze. Soon Koma understood the structure of the storehouse and drew a map of it. The map is a two-dimensional grid of $N \cdot K$ rows and M columns. Each square of the grid contains either an obstacle or a honey pot. An amazing property of the map is that it actually consists of K identical copies of the same $N \times M$ grid, placed one below the other!

The top side of each of the M columns of the grid is an entrance, and the bottom side of each column is an exit. Koma decides to take a tour in the storehouse obeying the following rules:

1. Koma enters the storehouse through one of the entrances, and leaves through one of the exits.
2. Koma can move from one square to another if these squares share a side.
3. At any point of time, Koma can occupy a square only if this square contains no obstacle.
4. Koma can visit each square at most once.
5. Koma can eat from each honey pot he visits.

Koma wants to visit as much honey pots as possible. For each possible entrance i and each possible exit j, compute two integers: the maximum number of honey pots he can visit when he enters through i-th entrance and leaves through j-th exit and the number of such optimal tours. As the number of tours can be rather large, find it modulo $10^9 + 7$.

Input

The first line of input contains three integers N, M and K ($1 \le N \le 5$, $2 \le M \le 7$, $2 \le N \cdot M \le 25$, $1 \le K \le 10^9$).

After that, there are N lines each of which contains M characters. They describe the repeating pattern of the grid. A character '`.`' means a square containing a honey pot, and a character '`X`' means a square containing an obstacle.

Output

Print two $M \times M$ matrices, one after another. The first matrix must contain the maximum number of honey pots Koma can visit. The second matrix must contain the number of possible ways to visit the maximum number of honey pots, taken modulo $10^9 + 7$. In each of the matrices, j-th number on i-th line corresponds to the case where Koma enters through i-th entrance and leaves through j-th exit.

Example

standard input	standard output	full grid

2 2 3
..
.X
	6 0
	7 0
	1 0
	1 0
		..
		.X
		..
		.X
		..
		.X

Problem 23. Catch Me If You Can

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

This is an interactive problem.

Carl Hanratty is a dour FBI agent usually pursuing criminals in the financial sector, but he also likes to play his Pokemon GO game in a park during his time off.

He still plays an old school version of the game with a working pokemon radar. This radar shows one, two or three paws telling him how far the pokemon is from the current player's location. Three paws means that the pokemon is not farther than $3 \cdot r$ meters from the player, two paws mean that it is not farther than $2 \cdot r$ meters, and one paw means not farther than r meters. It is known that the value of r is a real number between 1 and 100 meters, inclusive, but the exact value of r is unknown to Carl.

Since Carl is a very good FBI agent and always strives for excellence, he wants to improve his pokemon catching practice by utilizing a secret FBI analytical task force. You are a part of that analytical team. You will receive field information from Carl, and then direct his movements in order to find and catch the pokemon.

You may assume all the movements are happening on a two-dimensional plane. Carl is located at the point with coordinates $(0, 0)$, and he just saw the pokemon on the radar for the first time (with three paws). This means Carl is initially located $3 \cdot r$ meters away from the pokemon. To communicate with Carl, you are allowed to perform no more than 5 iterations of the following process:

1. You transmit to Carl an angle in degrees: the direction in which he should move. The angle might be any real value from 0 to 360 inclusive.
2. Carl goes straight in this direction until the radar indication changes. This happens if he catches the pokemon (then the game stops), reaches the radar zone that is closer to the pokemon (the number of paws on the radar decreases by 1) or reaches the radar zone that is farther from the pokemon (or pokemon disappears from the radar at all). In the last case, Carl immediately makes a step back to the previous three-paw zone, as he doesn't want to move in a direction which is definitely wrong.
3. Carl tells you the exact distance he traveled and the current number of paws p ($1 \leq p \leq 3$) of the radar zone. It means the current distance to the pokemon is $p \cdot r$. Note that the communication counts even if Carl returned a distance of 0 which means that walking any positive distance in the given direction will immediately result in radar indication change.

During your 5-th communication with Carl, his behavior is a bit different. As he knows he won't be able to get any instructions from you anymore, he ignores all the radar zones and moves straight in the direction you gave him until either he finds the pokemon or it totally disappears from the radar.

It is guaranteed that the pokemon location is fixed and does not change during each session of the game. It is also guaranteed that the unknown real value r lies between 1 and 100, inclusive.

Interaction Protocol

You have at most 5 communication attempts. To start the next attempt, your program must provide a direction to Carl by printing one real value a ($0 \leq a \leq 360$) on a separate line. After that, your program gets the result on the next line of the standard input.

If at any moment of time Carl is within $r/10$ from the actual pokemon location, he instantly catches it and returns you one final word "`Gotcha!`" (without quotes).

Otherwise, if you run out of communication sessions with Carl and he was unable to find the pokemon during the last move, he sends you one final line "`The pokemon ran away!`" (without quotes).

In all other cases, Carl transmits you the traveled distance d with at least ten digits after the decimal point and the integer number of paws p ($1 \le p \le 3$) of the new radar zone. These two numbers are separated by one space character.

Your communication with Carl must stop when the pokemon is caught or ran away.

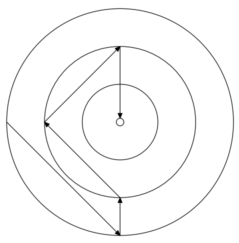
Examples

standard input	standard output
	90
0.0000000000 3	
	0
3.0000000000 2	
	270
0.0000000000 2	
	90
0.0000000000 2	
	0
Gotcha!	
	315
14.1421356237 3	
	90
3.3333333333 2	
	135
9.4280904158 2	
	45
9.4280904158 2	
	270
Gotcha!	

Note

In the first sample input, pokemon is sitting at point $(9, 0)$.

In the second sample input, pokemon is sitting at point $(10, 0)$.



The pipe from your program to the interactor program and the pipe back have limited size. Your program must read from the standard input to avoid deadlock. Deadlock condition is reported as "`Time Limit Exceeded`".

To flush the standard output stream, use the following statements:

In C, use `fflush(stdout)`;

In C++, use `cout.flush()`;

In Java, use `System.out.flush()`;

In Python, use `sys.stdout.flush()`.

If your program receives an `EOF` (end-of-file) condition on the standard input, it MUST exit immediately with exit code 0. Failure to comply with this requirement may result in "`Time Limit Exceeded`" error.

Problem 29. Demolition Time

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Darkwing Duck is in trouble again! As soon as he returned to his town, it was announced that Negaduck has been attacking the place. As a proper villain, he is going to rob some set of **consecutively** placed buildings and escape with the loot.

The panorama of the town can be represented as a string s where each character represents the shape of the corresponding building. Different buildings may have the same shape, and hence are represented in the string by equal characters. Negaduck has picked some string p as his robbery plan. Fortunately, this string p has become known to the police. Now they want to find all the buildings which are in danger, but there is one complication on the way.

By the order of the new Mayor, some of the buildings in the town should be demolished. They are demolished one after another **in the order they appear** in the string s: from left to right. This demolition plan is well known in the city.

The robbery can take place anytime: before all planned demolitions, after them, or between the demolition of any two buildings which follow one another in the demolition plan. However, no demolitions can happen during the robbery.

Negaduck robs buildings strictly according to his robbery plan from left to right. He does not skip any buildings except for demolished ones. He can execute only the whole plan. Nevertheless, there can exist a lot of subsets of buildings which are in danger of being robbed.

Formally, consider string t we get from s by removing the characters corresponding to already demolished buildings. Note that string t is initially equal to s but changes after each demolition. Negaduck can rob a subset of buildings if there exists a moment of time when this subset forms a substring of t, and this substring is equal to string p.

Your task is to find the exact amount of subsets which are in danger of being robbed.

Input

The first line of input contains a non-empty string s consisting of lowercase and uppercase English letters, digits, characters '`,`', '`!`', '`_`', '`.`' and '`-`' representing the buildings in the town from left to right. The length of this string doesn't exceed 1 000 000.

The second line of input contains a string p consisting of lowercase and uppercase English letters, digits, characters '`,`', '`!`', '`_`', '`.`' and '`-`' representing the Negaduck's plan from left to right. The length of this string doesn't exceed 1 000 000.

Note that uppercase and lowercase English letters are considered different.

The third line of input contains one integer m ($0 \le m \le |s|$) — the length of the Mayor's demolition plan.

The next line contains an **increasing** sequence of integers $x_1, x_2, \ldots, x_m$ ($1 \le x_1 < \ldots < x_m \le |s|$) — the indices of buildings in order they should be demolished.

Output

Output one integer: the number of different subsets of buildings that can be robbed according to Negaduck's plan.

Example

standard input	standard output
aabbcc abc 3 2 4 5	2

Note

The answer for the given sample is 2 because the following possible sequences of buildings can be robbed: 1 3 5 (after two building demolitions) and 1 3 6 (after all three building demolitions).

Problem 31. Votter and Paul De Mort

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

Young wizard Harry Votter is fighting the evil Paul De Mort. When Paul De Mort detected that his magical power is critically low, he casted the spell which created $N-1$ of his copies. Any two of N Paul De Morts are connected by the forcefield. That forcefield is a segment that connects these two copies (so $N(N-1)/2$ forcefields are created). For purpose of this problem, Paul De Mort and his copies are treated as points.

Harry can cast the spell «Magic Hexagram», which creates the L-Hexagram with the center at the origin. L-hexagram is the compound of two equilateral triangles such as the intersection of those triangles is a regular hexagon with side L. If Harry can select the rotation of Hexagram to completely cover all the forcefield segments, then Paul De Mort will be destroyed. If with any position of the hexagram atleast one of the forcefield segments is not covered completely, then Harry must change his plan and cast some other spell.

Given the value of L and coordinates of Paul De Mort and his copies, check if Harry can destroy Paul De Mort or he must change his plans.

Input

First line of the input contains one integer T $(1 \le T \le 1000)$ — number of test cases.

First line of each test case contains two integers N and L — number of Paul De Mort copies (including himself) $(3 \le N \le 1000)$ and the parameter of the spell L $(1 \le L \le 2 \cdot 10^6)$.

Then N lines follow, each containing two integers x_i and y_i, representing the position of one Paul De Mort's copies $(-2 \cdot 10^6 \le x_i, y_i \le 2 \cdot 10^6)$.

It is guaranteed that sum of all N in the input file does not exceeds $3.5 \cdot 10^5$.

Output

For each case, print at the separate line "`Cast`", if Harry can find the rotation of a Hexagram, which completely covers all forcefield segments, or "`Change`" otherwise.

Example

<table>
<tr><th>standard input</th><th>standard output</th></tr>
<tr><td><pre>2
4 10575
2462 9588
-16608 4264
-2462 -9588
16608 -4264
4 12497
-733 17594
-16629 -5795
733 -17594
16629 5795</pre></td><td><pre>Cast
Change</pre></td></tr>
</table>

Problem 37. Invisible Integers

Input file:	*stanard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

Invisible integers is a simple game where the player tries to guess a hidden sequence of integers 1 through 9 given a number of *hints*. Each hint is a sequence of *distinct* integers generated as follows:

- An arbitrary starting position is chosen in the hidden sequence.
- An arbitrary direction is chosen — either left or right.
- Starting from the chosen position we traverse all the integers of the hidden sequence in the chosen direction. We append the traversed integers to the end of the hint while skipping over the integers already in the hint.

Find the length of the shortest hidden sequence consistent with the given hints.

Input

The first line contains an integer n ($1 \leq n \leq 10$) — the number of hints. Each of the following n lines contains one hint. Each hint is a sequence of at least 1 and at most 9 distinct space-separated integers between 1 and 9 inclusive terminated with the integer 0.

Output

If there is no solution output -1. Otherwise, output a single integer — the length of the shortest hidden sequence consistent with the given hints.

Example

stanard input	standard output

5
1 2 0
3 4 0
1 4 3 0
3 1 4 2 0
1 2 4 3 0
	7

3
1 2 0
2 3 0
3 4 0
	-1

Note

In the first example, (1, 2, 1, 4, 1, 3, 4) is one sequence of minimal length consistent with the given hints:

- Hint (1, 2) can be obtained by starting from the 3rd element and heading left.
- Hint (3, 4) can be obtained by starting from the 6th element and heading right.
- Hint (1, 4, 3) can be obtained by starting from the 3rd element and heading right.
- Hint (3, 1, 4, 2) can be obtained by starting from the 6th element and heading left.
- Hint (1, 2, 4, 3) can be obtained by starting from the 1st element and heading right.

Problem 41. Indiana Jones and the Uniform Cave

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

Indiana Jones has stuck in the Uniform Cave. There are many round chambers in the cave, and all of them are indistinguishable from each other. Each chamber has the same number of one-way passages evenly distributed along the chamber's wall. Passages are indistinguishable from each other, too. The Cave is magical. All passages lead to other chambers or to the same one. However, the last passage, after all passages are visited, leads to the treasure. Even the exact number of chambers is a mystery. It is known that each chamber is reachable from each other chamber using the passages.

Dr. Jones noticed that each chamber has a stone in the center. He decided to use these stones to mark chambers and passages. A stone can be placed to the left or to the right of one of the passages. When Indiana Jones enters the chamber all that he can observe is the location of the stone in the chamber. He can move the stone to the desired location and take any passage leading out of the chamber.

Your task is to help Indiana Jones to visit every passage in the Uniform Cave and find the treasure.

Interaction Protocol

First, the testing system writes the integer m — the number of passages in each chamber ($2 \leq m \leq 20$).

Dr. Jones enters the chamber and sees, in the next line, where the stone is placed: either in the "`center`" of the chamber or to the "`left`", or to the "`right`" of some passage. On the first visit to the chamber, the stone is in the center.

Your solution shall output his actions: the number and the side of the passage to place the stone to, and the number of the passage to take. Both numbers are relative to the passage marked by the stone, counting clockwise from 0 to $m - 1$. If the stone is in the center of the chamber, the origin is random.

For example, "`3 left 1`" tells that Dr. Jones moves the stone three passages clockwise and places it to the left of the passage, then he takes the passage to the right of the initial stone position.

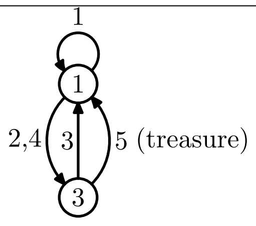
After each move testing system tells either the location of the stone in the next chamber or "`treasure`", if Indiana Jones had found it. The testing system writes "`treasure`" when all the passages are visited.

If Dr. Jones does not find the treasure room after 20 000 passages are taken, he starves to death, and your solution receives the "Wrong Answer" outcome. You also receive this outcome if your solution terminates before all passages are taken.

The total number of chambers in the cave is unknown, but you may assume that it does not exceed 20, and that each chamber is reachable from every other chamber.

Example

standard input	standard output	illustration

`2`
`center`
`left`
`center`
`left`
`right`
`treasure`
	`0 left 0`
	`1 left 1`
	`1 right 0`
	`0 left 0`
	`1 right 0`
		

Dr. Jones enters the example cave and sees that the stone in the first chamber is in the center. He marks the chamber by placing the stone to the left of some passage and takes it. He sees the chamber where the stone is to the left of the passage, so he is in the first chamber again. He moves the stone clockwise and takes the passage marked by it. This passage leads to the second chamber. He marks it by placing the stone to the right of some passage and takes another one. He is in the first chamber again, so he returns to the second chamber and takes the remaining passage. This passage leads to the treasure.

Problem 43. C-plus-minus

Input file:	`cpm.in`
Output file:	`cpm.out`
Time limit:	5 seconds
Memory limit:	512 mebibytes

Four pages long statement?!

Inexperienced participant

It's an implementation problem, sit down and code

Experienced participant who did not read the statement

How do I code that?

Experienced participant who did read the statement

Modern IDEs (integrated development environments) do a lot of stuff. Say, instantly suggest functions and variables based on types, show relevant documentation snippets, detect mistakes, rearrange code according to the style guide, consume all available RAM... In this problem you have to implement small piece of IDE for the $C\pm$ language.

In the beginning you have $C\pm$ program's source code called S_0. Your program should reformat the code according to rules below with adding and removing of spaces and newlines, yielding program F_0, which you have to print.

Afterwards you should perform m queries which change the source code. Let's denote the source code after performing i-th query ($1 \le i \le m$) as S_i. For each S_i we can yield a reformatted source code F_i. For each modification query you should print one integer: $L(F_i)$, where $L(x)$ stands for number of lines in source code x (see below for precise definition).

There are three types of queries:

- "`add` p @t@" query inserts string t into source S_{i-1}, starting at character number p (here $|t|$ stands for the length of t):

$$S_i = S_{i-1,1}S_{i-1,2}\dots S_{i-1,p-1}t_1t_2\dots t_{|t|-1}t_{|t|}S_{i-1,p}S_{i-1,p+1}\dots S_{i-1,|S_{i-1}|}$$

- "`del` p l" query removes a string of length l from S_{i-1}, starting at character number p:

$$S_i = S_{i-1,1}S_{i-1,2}\dots S_{i-1,p-1}S_{i-1,p+l}S_{i-1,p+l+1}\dots S_{i-1,|S_{i-1}|}$$

- "`get` r s k" query does not affect the source ($S_i = S_{i-1}$), but you should print a substring of source F_i. The desired substring is characterized with a line number r, number of the first desired character in that line s and length of the substring k.

  If F_i has less than r lines, we consider an empty string to be the answer for the `get` query. If r-th line in F_i does not have some characters with numbers $s, s+1, \dots, s+k-1$, we consider these missing characters to be equal to "#" (octothorpe, ASCII code is 35).

It's not guaranteed that $S_0, S_1, \ldots, S_n$ are correct $C\pm$ programs.

However, formatting rules described below can be applied to arbitrary sources.

Before defining formatting rules we'll introduce several supporting definitions:

- *Identifier* is a maximal by inclusion sequence of Latin letters, digits, square brackets "`[`" and "`]`" (ASCII codes 91 and 93), dots "`.`" (ASCII code 46) and underscores "`_`" (ASCII code 95). Some correct $C\pm$ identifiers: `hi`, `hello_world`, `bob[er9]5`, `12c.hairs`, `12_3`.
- *Binary operator* is one of the following characters:

Character	ASCII code
+	43
-	45
*	42
/	47
<	60
>	62
=	61
&	38
|	124

- *$C\pm$ allows the following characters*: parts of identifiers, binary operators, round brackets ("`(`", ASCII code 40 and "`)`", ASCII code 41), curly brackets ("`{`", ASCII code 123 and "`}`", ASCII code 125), comma "`,`" (ASCII code 44), semicolon "`;`" (ASCII code 59), space (ASCII code 32) and new line character (ASCII code 10, typically denoted by "\n" in modern programming languages). It's guaranteed that all sources $S_0, \ldots, S_n$ contain allowed characters only.
- *Lexem* is either identifier or any other allowed character, except for space and new line character. For example, source `hello+ world` contains three lexems: `hello`, `+` and `world`.
- *R-balance of brackets* (round or curly) in a some string S is the result of the following pseudocode (it counts balance for round brackets, code for curly brackets is similar):

```
B <- 0
for each C in S do {
  if C is "(" then B <- B + 1
  if C is ")" then B <- B - 1
  if B < 0 then B <- 0
  if B > R then B <- R
}
return B
```

  Not strictly speaking, this code calculates difference between number of opening and closing brackets of a specific type in S, but if this balance "gets beyond" $[0, R]$ segment, then it's "truncated" (R is either positive integer or ∞). For example, string "`{{{(){{}})}}(`" has ∞-balance of both round and curly brackets equal to 1, and string "`((((()`" has 4-balance of round brackets equal to 3.
- Position p is said to be *inside round brackets* if a substring t which ends at p and starts right after last curly bracket before p (or in the beginning of the source if there are no curly brackets before p), has 4-*balance of round brackets* strictly greater than zero. For example, if $t =$`(( ((( )))`, then p is *inside round brackets* (as 4-balance is 1), and if $t =$`(( ((( ))) )`, it is not (as 4-balance is 0).

Formatting is performed as follows:

1. All spaces and new line characters are removed from the source. Neighboring lexems are **not** considered as a single lexem after this step.

2. New line characters are added in the following places (unless empty lines appear):
   - After opening curly bracket “`{`”.
   - Before closing curly bracket “`}`”.
   - After closing curly bracket “`}`”, unless it's immediately followed by either “`else`” identifier or semicolon.
   - After semicolon “`;`”, unless it is located *inside round brackets*.
   - In the very end of the source.

   Line in the resulting program is defined as a sequence of characters ending with new line character. So, $L(x)$ is defined as the number of new line characters.

3. Spaces are added in each line between neighboring lexems according to the table below (plus on the intersection of row A and column B means adding of space between A and B):

	Identifier	Binary operator	`,`	`{`	`(`	`)`	`;`
Identifier	+	+	–	+	–	–	–
Binary operator	+	–	+	+	+	+	–
`,`	+	+	–	+	+	–	+
`}`	+	+	–	+	+	–	–
`(`	–	+	–	–	–	–	–
`)`	+	+	–	+	+	–	–
`;`	+	+	–	+	+	–	+

   Note that opening curly bracket `{` is always the last character in the line. Similarly, `}` is always the first character in the line.

4. If one of the identifiers `if`, `for`, `while`, `do` is immediately followed by opening round bracket, an additional space is added between them.

5. Afterwards an *indent* is calculated for each line. *Indent* is equal to ∞-balance of curly brackets in all previous lines.

6. Spaces are prepended to each line in the amount of doubled *indent* of the corresponding line.

Input

First lines of the input file contain original source code S_0 in the format `@`S_0`@` (at sign has ASCII code of 64). S_0 contains *allowed* characters only ($1 \le |S_0| \le 10^5$). Note that S_0 is not allowed to contain at sign, although it can contain spaces and/or new line characters. The second at sign is immediately followed by new line character.

The next line contains a single integer m, the number of queries ($0 \le m \le 10^5$). Next lines contain queries separated by at least one new line character, and i-th query is formatted as follows:

- “`add` p `@`t`@`”: add non-empty string t into current source, starting at position p ($1 \le p \le |S_{i-1}| + 1$). It's guaranteed that t contains *allowed* characters only. In particular, t may contain spaces and/or new line characters. The second at sign is immediately followed by new line character. It's guaranteed that sum of all $|t|$ does not exceed 10^5.

- “`del` p k”: erase a substring of length k from current source, starting at position p ($1 \le p \le p + k - 1 \le |S_{i-1}|$).

- “`get` r s k”: print k characters of line number r of current source, starting at position s ($1 \le r, s, k \le 10^5$). It's guaranteed that sum of all k does not exceed 10^5.

Please see example for better understanding of input format.

Output

At the beginning of the output file, print one integer $L(F_0)$: number of lines in the formatted source F_0. Afterwards print another space, at sign, $\min(10^5, |F_0|)$ characters of F_0, followed by another at sign and new line character.

Next, print the answer of i-th query on a separate line:

- For `add` and `del` queries: print number of lines in the source after corresponding change ($L(F_i)$).
- For `get` queries: print string `@`t`@` where t is the requested substring of length l of the formatted source F_i. If there is no line of number r in F_i (that is, $r > L(F_i)$), then you should print empty string instead of t. If $r \le L(F_i)$, but some of requested characters do not exist in F_i, you should print "`#`" character (ASCII code 35) instead of them. We consider new line character to be not included in t.

Please see example for better understanding of output format. Follow the format as close as possible: your answer must be identical to the required answer.

Example

cpm.in

```
@if (a) { foo } else { bar; };
for (int a = 0; a< 5; a+=1)
{ some strange(command, wow); }}
else int foo bar {
  {(hi; strange; world)}

   (hi; strange; world() {there;}
}@
14
get 3 2 10
get 100 1 1
add 32 @NEW
LINE@
get 6 1 30
del 32 8
add 18 @se if @
get 2 1 10
get 3 1 10
get 4 1 10
del 20 1
get 2 1 10
get 3 1 10
get 4 1 10
get 5 1 10
```

cpm.out

```
16 @if (a) {
  foo
} else {
  bar;
};
for (int a = 0; a < 5; a += 1) {
  some strange(command, wow);
}
} else int foo bar {
  {
    (hi; strange; world)
  }
  (hi; strange; world() {
    there;
  }
}
@
@ else {###@
@@
16
@fNEW LINEor(int a = 0; a < 5; @
16
16
@  foo#####@
@} else if @
@  bar;####@
17
@  foo#####@
@}#########@
@elseif se @
@  bar;####@
```