Problem A. Mixed Messages

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

Nikita received some messages today. One of the messages was the code word "`spbsu`". Before and after the code word, there may have been any number of other messages as well. The other messages are arbitrary strings of lowercase English letters.

All the messages were sent via a secret channel, one by one. So, the string in the channel initially was a concatenation of all the messages.

However, being secret, the channel may introduce noise: different messages may interfere with each other. Formally, the noise comes in form of *swaps*. In each swap, the channel selects and exchanges two adjacent letters in the string **that initially belonged to different messages**. For letters of any particular message, the relative order is preserved.

After all swaps, the resulting string is received by Nikita. Given the resulting string, find the minimum possible number of swaps made by the channel.

Input

The first line contains a single integer n: the number of received characters ($5 \le n \le 10^5$).

The next line contains a string s consisting of n lowercase English letters: the resulting string received by Nikita. It is guaranteed that this string is the result of the process described above.

Output

Output a single integer: the answer to the problem.

Examples

standard input	*standard output*
6 spbssu	1
15 spongebaseurban	11

Problem B. I Flipped The Calendar...

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

While flipping through the calendar, Nikolai wondered: how many rows are in the calendar for a specific year?

The calendar consists of 12 sheets, each corresponding to a month from January to December. Each sheet lists all the days of the respective month. The days on each sheet are arranged in rows by week: the days of one week are in one row, the days of different weeks are in different rows. In this calendar, the week starts on Monday.

For example, if a month has 31 days and the first day of the month is Sunday (as in January 2023), then there will be six rows on the calendar sheet for that month:

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Remember that in a leap year, February has 29 days, and in a non-leap year, it has 28 days. A year is considered a leap year if its number is divisible by 400 or divisible by 4 but not by 100. For example, 2000, 2004, and 2040 are leap years, while 1900, 1982, and 2039 are not.

Input

The first line contains the year number y $(1970 \le y \le 2037)$.

Output

Output the number of rows in the calendar for the given year.

Example

standard input	*standard output*
2023	63

Problem C. Many Many Cycles

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

Consider an undirected graph G. Find the maximal number d such that the lengths of all simple cycles are divisible by d. If there is no such number, output 0.

Input

The first line contains two integers n and m: the number of vertices and edges ($1 \le n \le 5000$, $0 \le m \le 10\,000$). Each of the next m lines contains three integers a, b, and c, which mean that there is a bidirectional edge between vertices a and b with length c ($1 \le a, b \le n$, $1 \le c \le 10^9$). It is guaranteed that the graph doesn't contain loops or multiple edges.

Output

Print one integer: the answer to the problem.

Examples

standard input	*standard output*

4 4
1 2 1
2 3 1
3 4 1
4 1 1
	4

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

Problem D. Bishops

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

A chess bishop attacks every square that shares a diagonal with it.

Place the maximum number of bishops on an $n \times m$ chessboard in such a way that none of them attack each other.

Input

The first line contains two integers n and m: the dimensions of the chessboard ($1 \le n, m \le 10^5 + 1$).

Output

On the first line, print an integer k: the maximum possible number of bishops on an $n \times m$ chessboard such that they don't attack each other. On each of the next k lines, print two integers: the coordinates of bishops. The first coordinate should be in the range $[1, n]$, and the second in the range $[1, m]$. If there are several possible answers, print any one of them.

Examples

standard input	*standard output*

2 5	6
	2 5
	1 5
	2 3
	1 1
	1 3
	2 1

5 5	8
	1 1
	1 2
	5 4
	1 3
	5 3
	1 4
	5 2
	1 5

Problem E. Fischer's Chess Guessing Game

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

This is an interactive problem.

Fischer's Chess is a game in which the starting position is randomly chosen from 960 positions. The starting position satisfies the following requirements:

- The white pieces are shuffled (the king, queen, two rooks, two bishops, and two knights are placed on the first rank).
- The black pieces are placed on the eighth rank in the same order as the white pieces.
- The king is placed between the two rooks (there may be other pieces between the king and rooks). This is necessary for castling rule.
- The white bishops are placed on opposite-colored squares. In other words, one bishop is on a square of even file, and the other is on a square of odd file.

Jill has chosen one of the starting positions. Jack's task is to guess it. Jack places any legal starting position on the chessboard, and Jill tells him how many of the white pieces he has placed correctly. Your task is to help Jack guess the position by placing no more than six positions.

Interaction Protocol

The interaction consists of no more than 100 games. At the beginning of each game, you need to read a line from the input containing the word "GAME" and an integer n corresponding to the game number (starting from 1).

When attempting to guess the position, output a string of 8 characters representing the arrangement of the white pieces. The king is denoted by character "K", the queen by "Q", the rook by "R", the bishop by "B", and the knight by "N". In response, read a line from the input containing an integer from 0 to 8: the number of correctly placed white pieces. The game ends when you read number 8 (all pieces are placed correctly) or the limit on the number of guesses is exceeded (in this case, the outcome will be "Wrong Answer" if the solution terminates immediately, or it can be any non-OK if the solution tries to read more input).

At the end of the interaction, read a line from the input containing the word "END".

After outputting each position, don't forget to print the newline character and flush the correct output buffer, or the outcome will be "Idleness Limit Exceeded". To flush the buffer, one can call, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

In each game, the permutation is fixed in advance and does not change.

Example

standard input	*standard output*
GAME 1	
	RQKBBNRN
4	
	RQKNBRNB
3	
	RKRNBBQN
5	
	RKRBBQNN
8	
END	

Problem F. Forward-Capturing Pawns

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

In many combinatorial games, even a minor change of the rules can completely alter the "landscape" of the game. Chess is no exception. Normally, the pawn is the only chess piece that doesn't capture in the same way as it moves: the pawns move forward, but capture diagonally. But what would happen if there was no such distinction?

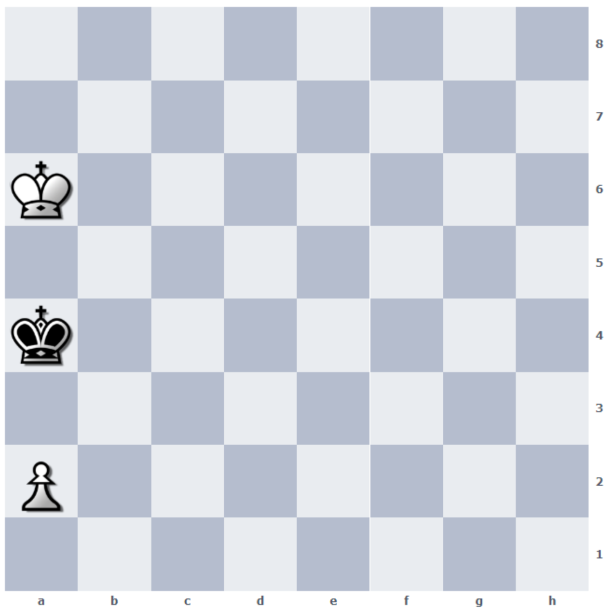
For sure, the resulting game would feel vastly different from the chess we know. There would be no closed pawn structures, because pawns could not be blocked at all. In fact, in our "new" chess, there is only one way to stop an advancing pawn: to capture it. Therefore, there are no "closed" (in the classical sense) positions. Nor is there any way to create a pair of doubled pawns.

It is clear that such a change would completely alter the way we play chess at every stage of the game: the opening, the middlegame and the endgame. Your task is to investigate how the change impacts the simplest part of the endgame theory: the king and pawn versus king endgames.

All other rules remain the same. You only have to consider *reasonable* positions. A position is reasonable if and only if all the following conditions hold:

1. All three pieces (White king, White pawn, Black king) are in pairwise different squares.
2. White pawn is neither in the 1-st nor in the 8-th row.
3. If it is White's turn, then Black king is not in check. If it is Black's turn, then White king is not in check. In particular, kings can't reside in adjacent (either by side or by corner) squares.

There are two important things to note here. Firstly, **if the pawn resides in the 2-nd row, then it controls two squares in front of itself**. That is, the pawn captures *exactly* the same way as it moves. For example, a `c2` pawn controls both squares `c3` and `c4`. Consider the following position:



Only reasonable if it is Black's turn. Black are in check.

With White to move, it is not reasonable, because the Black king is in check. With Black to move, it is reasonable and Black has only two valid moves: `Kb3` and `Kb4`, because `a3` is controlled by the White pawn and `a5` and `b5` are controlled by the White king.

There is a second important thing to note. While the above position is reasonable (with Black to move), it is not legal in the usual sense (reachable from the starting position by a legal sequence of moves). If you think about it, it becomes clear that there is no possible previous move White could have made. All legal positions are reasonable, but not all reasonable positions are legal. Informally, reasonable positions are those that "look legal at first glance". **You are specifically asked to solve the problem for reasonable positions**, so some positions in the input can be illegal in the usual sense.

As mentioned before, all the usual rules of the game remain the same. If the pawn is in the 2-nd row, it may choose to move two squares forward, if both those squares are empty (but, of course, it **can't** jump over the White king). When the pawn reaches the 8-th row, it can be promoted to a queen, a rook, a bishop or a knight. Stalemate and three-fold repetition are still draws. You may assume that there is no 50-move rule (it can be proven that adding/removing the 50-move rule never changes the outcome of the game).

Input

The first line of the input contains a single integer t: the number of positions to consider ($t \geq 1$). The following t lines describe the positions.

Each position is described in the following way: the squares with the White king, the White pawn and the Black king (in this order), and whose turn is right now ("`w`" for White or "`b`" for Black). The squares are described in the standard way ("`a`"–"`h`" for columns and "`1`"–"`8`" for rows). The four tokens on the line are separated by single spaces. There are no extra spaces at the end of the line. Please refer to the sample input if in doubt.

All positions in the input are reasonable and pairwise distinct. The positions that differ only by the moving side are considered to be **different**.

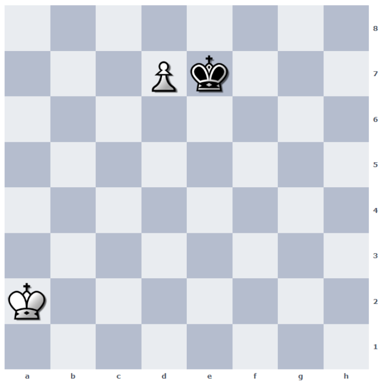
Output

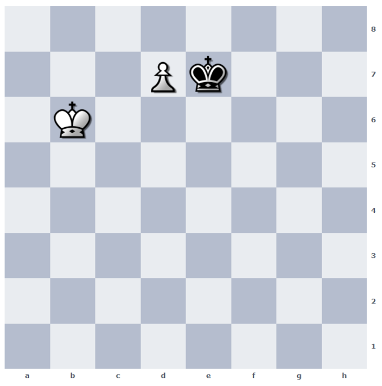
For each position in the input, print a single line: "`Win`" if White wins and "`Draw`" otherwise.

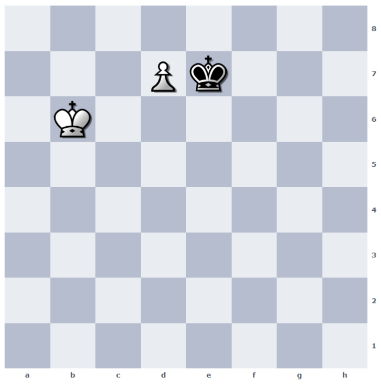
Example

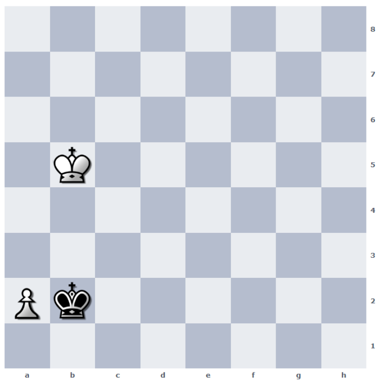
standard input	*standard output*
6 a2 d7 e7 w b6 d7 e7 b b6 d7 e7 w b5 a2 b2 w a6 a2 a4 b g6 g7 h8 b	Draw Draw Win Win Draw Draw

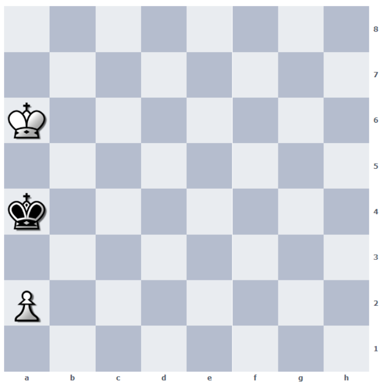
Note

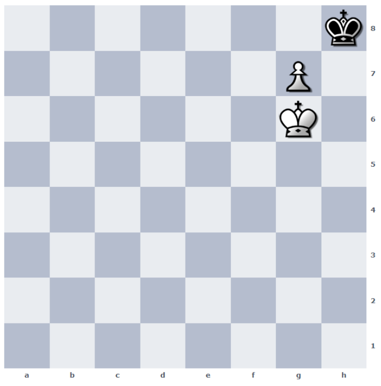

White to move, draw. White can promote, but Black will immediately capture the resulting piece.


Black to move, draw. Black can immediately capture the pawn.


White to move, win. White plays `Kc7` and safely promotes the pawn. Notice that the only difference between this and the previous position is the moving side.


White to move, win. White plays `a4`, using the right to move a pawn two squares forward if it is in the 2-nd row.


Black to move, draw. Black plays `Kb3` and captures tha pawn on the next move.


Black to move, draw. Black has no legal moves, but their king is not in check. Therefore, it is stalemate and the position is drawn by definition.

Problem G. Graph Cuts

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	1024 mebibytes

You are given an undirected graph without multiple edges or self-loops. You also have a set of its vertices U that is initially empty. Your task is to answer queries of the following form.

1. "`+` v". Add vertex v to U. It is guaranteed that $v \notin U$.
2. "`-` v". Remove vertex v from U. It is guaranteed that $v \in U$.
3. "`?`". Find an edge such that exactly one of its endpoints is in U and remove it from the graph, or determine that there are no such edges. If there are multiple edges that fulfill this property, you can choose any one of them.

Input

The first line contains two integers n and m: the numbers of vertices and edges in the graph correspondingly ($0 \le n, m \le 10^5$). Each of the next m lines contains two integers u and v: the endpoints of a bidirectional edge ($1 \le u, v \le n$). It is guaranteed that there are no multiple edges and no self-loops in the graph.

The next line contains a single integer q, the number of queries ($0 \le q \le 10^5$). The next q lines contain queries in the format described above ($1 \le v \le n$ in the queries).

Output

For each query of the third type, your program should either print a number of the found edge in the order it was presented in the input, or print 0 if such an edge does not exist.

Example

standard input	*standard output*
4 5 1 2 1 3 1 4 2 3 2 4 10 + 1 + 2 ? ? ? ? ? - 2 ? ?	5 4 3 2 0 1 0

Problem H. Very Sparse Table

Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	512 mebibytes

This is an interactive problem.

You are given a directed graph G on vertices numbered 0 to n. Initially, G contains exactly n edges of the form $v \to v + 1$. Your task is to add some edges to this graph in such a way that for every two vertices v, u $(v < u)$ there exists a directed path from v to u consisting of at most three edges. There are also two additional requirements you must meet:

1. You can add an edge $a \to c$ if and only if there exists such b that edges $a \to b$ and $b \to c$ are already present in G.
2. You can add at most $6 \cdot n$ edges in total.

Interaction Protocol

The interaction starts with the interactor printing the only integer n $(0 \le n \le 2^{16})$ on a single line. After that, your program should output k $(0 \le k \le 6 \cdot n)$: the number of edges you want to add to the graph. The next k lines should contain descriptions of the added edges in the order of their addition. Each edge should be described by three integers a, b, c, meaning that you add the edge $a \to c$, and that edges $a \to b$ and $b \to c$ are already present in G.

Do not forget to flush the output buffer after this (you can use `cout << flush;` in C++), otherwise, the solution will get "`Idleness Limit Exceeded`".

Then, the interactor prints the number of queries q $(0 \le q \le 2 \cdot 10^5)$ and q queries on the next q lines. Each query is described by two integers ℓ, r $(0 \le \ell < r \le n)$ for which your program should output a line containing a path in G of length at most three that starts in vertex ℓ and finishes in vertex r. The path should be printed as a sequence of all vertices it visits (including both endpoints) separated by spaces.

To get the next query, the solution **must** print the answer to the previous query, end the line with a newline and flush the output buffer. The queries **depend** on the edges you printed.

In the example below, there are **no** actual empty lines in input and output: they are only added to visually align the inputs and the respective outputs.

Example

standard input	*standard output*
9	7
	1 2 3
	0 1 3
	6 7 8
	6 8 9
	3 4 5
	4 5 6
	3 5 6
3	
1 8	1 3 6 8
2 4	2 3 4
0 5	0 1 3 5

Problem I. Password

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

After another leak of personal data, the administrator of Pochta.com decided to tighten the rules for employee passwords. Now, each employee's password must consist of exactly n characters, and non-letter characters must occur among every three consecutive characters. Additional restriction is that the non-letter character must be present in the center of the password: one center character if n is odd, or both characters closest to the center if n is even.

For example, for $n = 9$, the following passwords are valid: "`p4ss#or0s`", "`1a2b34CD5`". The password "`1234a56bc`" is not valid because the fifth character must be non-letter. The password "`9ASE#orkd`" is not valid because it contains three letters in a row.

For $n = 6$, the passwords "`ab23bc`" and "`5a428E`" are valid. The passwords "`111e11`" and "`4sy1um`" are not valid.

The employees now wonder: what is the minimum and maximum number of non-letter characters that can occur in a password of a given length? Help them figure this out.

Input

The first line contains an integer n: the length of the password ($1 \le n \le 1\,000\,000$).

Output

Output two integers separated by a space: the minimum and maximum number of non-letter characters in the password.

Examples

standard input	*standard output*
1	1 1
2	2 2
3	1 3

Problem J. Transport Pluses

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

Cambeet lives on a plane. He wants to travel from his home located at point (x_h, y_h) to the exhibition located at point (x_e, y_e). There are two ways to travel on the plane, and each consumes energy.

First, one can move along a segment between two points. The energy required for such travel is equal to the length of the segment: moving from point (x_a, y_a) to points (x_b, y_b) consumes $\sqrt{|x_\text{b} - x_\text{a}|^2 + |y_\text{b} - y_\text{a}|^2}$ units of energy.

Second, there are n *transport pluses* on the plane, numbered by integers from 1 to n. Plus i is centered at point (x_i, y_i) and connects all points with $x = x_i$ or $y = y_i$: one can instantly travel from any such point to any other such point, they just have to input the coordinates in a mobile app. Each use of any plus consumes t units of energy.

Cambeet can use any ways of travel in any order. Help him find a path that will require the minimum total amount of energy.

Input

The first line contains two integers n and t: the number of transport pluses and the energy consumed by every use of a plus $(0 \le n, t \le 100)$. The second line contains two integers x_h and y_h: the coordinates of Cambeet's home $(0 \le x_\text{h}, y_\text{h} \le 100)$. The third line contains two integers x_e and y_e: the coordinates of the exhibition $(0 \le x_\text{e}, y_\text{e} \le 100)$. Each of the next n lines contains two integers x_i and y_i: the coordinates of the center of i-th plus $(0 \le x_i, y_i \le 100)$.

All input data are integers. However, Cambeet can freely travel to points with any real coordinates.

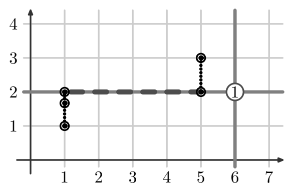
Output

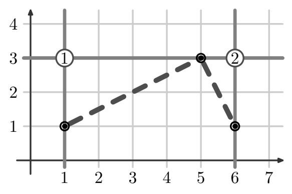
On the first line, print a real number: the total consumed energy. On the second line, print an integer k: the number of moves in the path $(0 \le k \le 10\,000)$. On each of the next k lines, print three integers p_j, x_j and y_j: the number of the plus being used and the coordinates of the next target. The value of p_j can be zero, which means traveling along a segment, or an integer from 1 to n, which means using a plus with such number. The coordinates can be any **real** numbers from 0 to 100. When using a plus, this plus must connect the current position and the next target. The path must end at (x_e, y_e).

You can print any path that requires the minimum total amount of energy. The answer will be considered correct if the total consumed energy differs from the minimum possible by at most 10^{-3}.

Examples

standard input	*standard output*	*illustration*

1 2
1 1
5 3
6 2
	4.000000
	4
	0 1 1.67
	0 1 2
	1 5 2
	0 5 3
		

2 1
1 1
6 1
1 3
6 3
	2.0
	2
	1 5 3
	2 6 1
		

Problem K. Poor Students

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

End of semester is coming, and it is a hard time for students. There are k courses and n students, and every student should pick exactly one course and pass the exam on it.

If student i picks exam j, the student's frustration will be $c_{i,j}$. The total frustration of students is the sum of their individual frustrations.

The teachers insist that, for each exam j, no more than a_j students can pick this exam. What is the minimum possible total frustration the students may get?

Input

The first line contains two integers n and k: the number of students and the number of exams ($1 \le n \le 50\,000$, $1 \le k \le 10$).

Then follow n lines. In i-th of these lines, there are k integers $c_{i,1}, c_{i,2}, \ldots, c_{i,k}$: the frustration of student i if they choose the exam $1, 2, \ldots, k$ ($1 \le c_{i,j} \le 10^9$).

The last line contains k integers $a_1, a_2, \ldots, a_k$: the maximum number of students that can pick exam $1, 2, \ldots, k$ ($0 \le a_j \le n$). It is guaranteed that $\sum a_j \ge n$.

Output

Print one integer: the minimum possible total frustration.

Examples

standard input	*standard output*

6 2
1 2
1 3
1 4
1 5
1 6
1 7
3 4
	12

3 3
1 2 3
2 4 6
6 5 4
1 1 1
	8

Problem L. “Memo” Game With a Hint

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

“Memo”, or “Memory”, is a game with cards. In this problem, the game uses 25 pictures, and each picture is printed on exactly two rectangular cards: so, there are 50 cards in total. Initially, all cards are randomly shuffled and dealt face down, so that the pictures are not visible. The cards occupy positions which are numbered by integers from 1 to 50.

Vasilisa trains her memory by playing “Memo”. She makes moves in the game. Each move consists of two actions. For the first action, Vasilisa picks a position with a card face down and turns it face up. For the second action, she picks another such position and also turns the card face up. If the two pictures on these cards are the same, they remain face up. Otherwise, they are both turned face down again, and Vasilisa gets a *miss*.

Vasilisa wins when all cards are face up. Her task is to win and get as little misses as possible.

Vasilisa has trained her memory well and devised a good strategy, and now, in each game, she makes approximately 14.83 misses on average. To make things more interesting, she called her sister Sasha to play together.

The backs of the cards are all the same, but they look differently when the card is *rotated* 180 degrees without turning it face up. Now Sasha and Vasilisa play as follows. Firstly, Sasha shuffles the cards and deals them face up. Then she turns them face down, but for each card, she can either rotate it 180 degrees, or not rotate. Finally, Vasilisa initially sees not only 50 cards with the same backs, but for each card, she sees whether it is rotated 180 degrees.

Devise a way for Sasha and Vasilisa to agree how the cards are rotated, so that Vasilisa can win with no more than 13.5 misses on average.

Interaction Protocol

This is an interactive problem. Additionally, in this problem, your solution will be run twice on each test. Each line of input is terminated by an end-of-line character.

First Run

During the first run, the solution acts for Sasha. The first line contains the word “`prepare`”. The second line contains an integer t, the number of test cases ($1 \le t \le 100$). Each of the next t lines describes the initial positions of the cards: 50 uppercase English letters “`A`”–“`Y`” without spaces. Equal pictures correspond to equal letters. Each letter occurs in the string exactly twice. It is guaranteed that, except the examples, all strings in each test are picked in advance, randomly and uniformly.

For each test case, print a line with 50 binary digits: each position should contain a one if the card is rotated 180 degrees, and a zero otherwise.

Technically, this run is interactive, but all the input is given at once.

Second Run

During the second run, the solution acts for Vasilisa. The first line contains the word “`play`”. The second line contains an integer t, the number of test cases which is the same as during the first run.

The jury program counts the total number of misses for all test cases. This number has to be at most 1350: for example, if there are exactly 100 test cases (and not less), it means that one test case can have at most 13.5 misses on average.

Each test case starts by a line containing 50 binary digits, which is the line that the solution printed during the first run. After that, the solution has to interactively print actions until all cards are face up.

To turn the card at position p (an integer from 1 to 50), print a line containing the number p. In response, the solution gets a line with two characters. This line is "`##`" (two hashes) if the move is invalid: for example, the position p is invalid, or the card at that position is already face up, or this move exceeds the maximum possible number of misses. In this case, solution has to terminate. Otherwise, the first character is an uppercase English letter "`A`"–"`Y`" at position p corresponding to the picture. The second character can be one of the following:

- "`.`" (dot) if it is the first action of a move;
- "`-`" (minus) if it is the second action of a move, and the pictures are different;
- "`+`" (plus) if it is the second action of a move, the pictures are the same, but Vasilisa did not yet win;
- "`!`" (exclamation mark) if it is the second action of a move, the pictures are the same, and this is the winning action.

After a response with an exclamation mark, the next line of input belongs to the next test case, or the interaction is terminated if there are no more test cases.

Do not forget to end the line and flush the output buffer immediately after printing each action: otherwise you will likely get the "`Idleness Limit Exceeded`" outcome.

When the solution gets the line "`##`", it can terminate immediately, in order to get the "`Wrong Answer`" outcome and not any other.

Examples

For each test, the input during the second run depends on the solution's output during the first run.

Below we show two runs of a certain solution on the first test. Empty lines are added only for readers' convenience: there will be no empty lines during a real run. To conserve space, almost all moves are replaced with "`(...)`".

standard input	*standard output*
`prepare`	
`2`	
`ABCDEFGHIJKLMNOPQRSTUVWXYABCDEFGHIJKLMNOPQRSTUVWXY`	
	`00000000000000000000000001111111111111111111111111`
`AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSSTTUUVVWWXXYY`	
	`11010011001111111111111111111111111111111111111010`

standard input	*standard output*
`play`	
`2`	
`00000000000000000000000001111111111111111111111111`	
	`1`
`A.`	
	`5`
`E-`	
	`26`
`A.`	
	`1`
`A+`	
`(...)`	`(...)`
	`25`
`Y.`	
	`50`
`Y!`	
`11010011001111111111111111111111111111111111111010`	
	`10`
`E.`	
	`9`
`E+`	
`(...)`	`(...)`
	`1`
`A.`	
	`2`
`A!`	

Problem M. Hardcore String Counting

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

You are given a non-empty string s of lowercase English letters. A string w of lowercase English letters is *good* if every proper prefix of w does not contain s as a substring, but w itself does.

Find the number of good strings of length m. Because this number can be very large, output it modulo prime number $998\,244\,353 = 2^{23} \cdot 119 + 1$.

Input

The first line of the input contains two integers: n, the length of s, and m, the length of strings you have to count ($1 \le n \le 10^5$, $n \le m \le 10^9$). The second line contains a string s consisting of n lowercase English letters.

Output

Output a single nonnegative integer: the number of good strings of length m modulo $998\,244\,353$.

Examples

standard input	*standard output*
6 7 aaaaaa	25
3 5 aba	675