Problem A
Bingo Ties

Photo by Samueles

Bingo is a game of chance played by a group of players. Each player has his or her own bingo card with a 5-by-5 grid of numbers. Each number appears at most once per card. The bingo caller calls out a sequence of randomly drawn numbers, and the players mark the numbers that appear on their card as they are called out. The winner is the player that completes a line of five marked numbers (horizontally, vertically or diagonally) on his or her card. The winning player then yells "bingo" and the game ends.

You have been volunteering with a local youth group and have been running bingo games for the children. They love bingo, but every time two or more kids yell "bingo" at the same time, a spirited "disagreement" breaks out. You've created a slightly modified version of bingo (in the hopes of introducing fewer ties): the cards are 5-by-5 grids with a number from 1 to 3 000 in each of the 25 cells, and a winner is only declared when a player has 5 numbers in a row. Note that in this new game, players **cannot win via columns or diagonals**.

Alas, these changes did not eliminate ties or the subsequent disagreements. To prevent further disagreements, you've decided to analyze the sets of cards to determine if there is any possibility that a tie (where two kids can yell bingo at the same time) can occur. Write a program that takes a collection of bingo cards and determines if there is any possible sequence of numbers that could be called so that the game ends, and a tie between two or more players occurs, when the last number in the sequence is called.

For example, consider the following two bingo cards:

3	29	45	56	68
1	19	43	50	72
11	25	40	49	61
9	23	31	58	63
4	27	42	54	71

14	23	39	59	63
8	17	35	55	61
15	26	42	53	71
10	25	31	57	64
6	20	44	52	68

Then this set of two cards could result in a tie if the sequence of numbers called was

40 61 64 10 57 49 11 31 25

This sequence would result in the card on the left completing the third row and the card on the right completing the fourth row when the number 25 is called.

Input

The first line of the input is an integer n ($2 \leq n \leq 100$), the number of bingo cards. After the first line are the n bingo cards, each separated from the next by a blank line of input.

Each bingo card consists of five lines of input. Each line consists of five integers in the range from 1 to 3 000. The numbers on each bingo card are unique.

Output

If no ties are possible between any two cards, output "`no ties`" on a single line. Otherwise, output the numbers of the lowest numbered pair of cards for which a tie could occur, where the cards are numbered from 1 to n in the order that they appear in the input. If multiple pairs of cards can tie, output the pair that is lexicographically smallest (with a smallest first card number, and then a smallest second card number).

Sample Input 1

```
2
3 29 45 56 68
1 19 43 50 72
11 25 40 49 61
9 23 31 58 63
4 27 42 54 71

14 23 39 59 63
8 17 35 55 61
15 26 42 53 71
10 25 31 57 64
6 20 44 52 68
```

Sample Output 1

```
1 2
```

Sample Input 2

```
2
2189 2127 1451 982 835
150 1130 779 1326 1149
2697 2960 315 534 2537
2750 1771 875 1702 430
300 2657 2827 983 947

886 738 2569 1107 2758
2795 173 1718 2294 1732
1188 2273 2489 1251 2224
431 1050 1764 1193 1566
1194 1561 162 1673 2411
```

Sample Output 2

```
no ties
```

Problem B
Das Blinkenlights

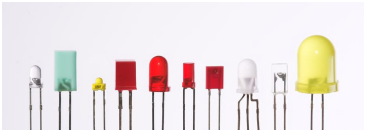


Photo by Afrank99

There are two lights that blink at regular intervals. When each one blinks, it turns on and then immediately back off; they don't toggle. They are both off at time $t = 0$. The first one blinks at $t = p, 2p, 3p, \ldots$ seconds; the second one blinks at $t = q, 2q, 3q, \ldots$ seconds. Once they start, they both keep blinking forever. It is *very* exciting to see them blink at the same time (on the same second). But your patience is going to run out eventually, in s seconds. Will they blink at same time between $t = 1$ and $t = s$ (inclusive)? Write a program that can answer this question, quick, before they start blinking again!

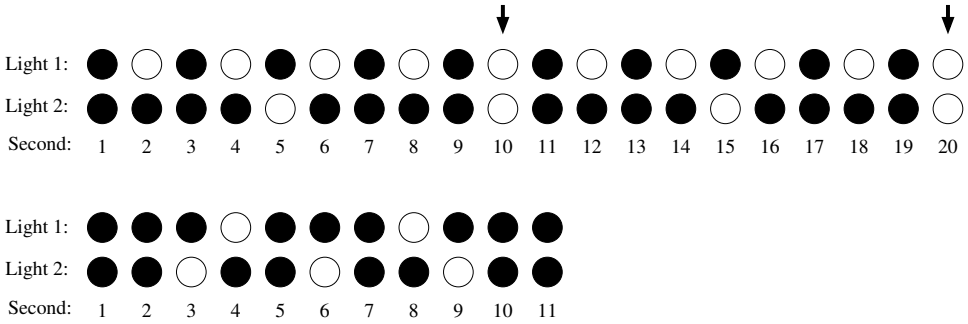


Figure B.1: Illustration of the sample inputs. A black circle means the light is off, a white circle means the light blinks at that second. The arrows above point out times when both lights blink.

Input

Input consists of one line containing three space-separated integers p, q, and s. The bounds are $1 \leq p, q \leq 100$ and $1 \leq s \leq 10\,000$. The first light blinks every p seconds, the second every q seconds. The value of s represents the maximum number of seconds to consider when determining if the two lights blink at the same time.

Output

Output `yes` if the two lights blink on the same second between time 1 and time s, or `no` otherwise.

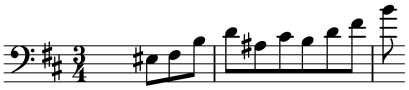
Sample Input 1	Sample Output 1
2 5 20	yes

Sample Input 2	Sample Output 2
4 3 11	no

This page is intentionally left blank.

Problem C
Ebony and Ivory

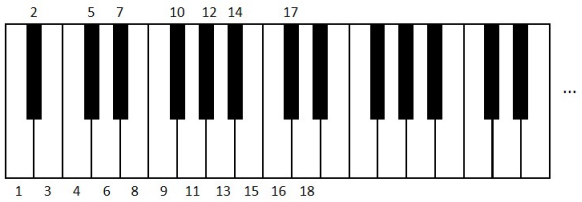
Piano fingering describes the process of assigning fingers of a hand to keys on the piano when playing a musical passage. Although professional musicians consider piano fingering an art rather than a science, students often search for finger assignments that make them easier to play.



A passage from Chopin's Scherzo No. 1.

The goal of this problem is to find an assignment that minimizes the ergonomic difficulty of playing a right-hand passage without chords, i.e., where one note is played at a time.

A standard piano keyboard has 52 white keys and 36 black keys for a total of 88 keys, which are numbered $1 \ldots 88$. These keys are interleaved as shown in Figure 3. Thus, in this numbering, keys $2, 5, 7, 10$, and 12 are black, as are any keys whose number is any multiple of 12 away from these keys (e.g., $14, 17, 19, \ldots$).



A passage is a sequence of notes (each of which corresponds to a key). The difficulty of playing a passage is the sum of the difficulties of playing each interval, i.e., pair of adjacent notes in the passage. A passage of length L thus consists of $L - 1$ intervals. An interval's difficulty depends on the following factors:

- The distance between the keys, which is counted in so-called 'half steps.' Stepping from key number i to $i + 1$, or from i to $i - 1$, is moving one half step. If an interval uses the same key twice in a row (zero half-steps), this key must be played with the same finger because the next note may need to be played connected, or *legato*. The difficulty of playing intervals in which a note is repeated is zero.
- Which finger is used to play the lower key and which is used to play the upper key. The lower key is the one with the smaller number.
- Whether the lower and upper keys are white or black.

Since these latter two difficulties can vary between players, we use ergonomic tables, which are indexed using pairs of fingers assigned to the lower and upper keys of an interval. The fingers of the right hand are numbered from 1 (thumb) to 5 (little finger). Since there are 4 possible color combinations for the lower and upper key (white/white, white/black, black/white, and black/black) there are 4 ergonomic tables that describe the transition difficulties. Consult the proper table based on the colors of the lower and upper keys in an interval. Playing an ascending interval has the same difficulty as playing the same

interval in descending order if the lower and upper keys are assigned to the same fingers. For example, playing the interval $(41, 43)$ with fingers $(1, 3)$ is as difficult as playing interval $(43, 41)$ with fingers $(3, 1)$ – in both cases, finger 1 is on key 41, and finger 3 is on key 43.

Input

The input consists of a single test case. The first line contains 5 positive integers w_w, w_b, b_w, b_b ($0 < w_w, w_b, b_w, b_b \leq 20$), and L ($1 \leq L \leq 10\,000$). Four ergonomic tables follow back-to-back, in the order white/white, white/black, black/white, and black/black, with w_w, w_b, b_w, b_b entries each. Each table entry is listed on a separate line containing 14 integers in the following format: f_l f_u h_1 h_2 h_3 ... h_{12}. h_i ($0 \leq h_i \leq 10$) is the relative difficulty of playing an interval of i half steps if finger f_l is used to play the lower key and finger f_u is used to play the upper key of the interval. f_l and f_u ($1 \leq f_l, f_u \leq 5$, $f_u \neq f_l$) denote fingers of the right hand.

The last line of the input contains L integers a_i ($1 \leq a_i \leq 88$, $|a_{i+1} - a_i| \leq 12$ for all i), which represent the sequence of notes that make up the passage. It is guaranteed that a suitable finger assignment exists.

The tables found in Sample Input 1 are due to Hart, Bosch, and Tsai.

Output

Output a single number, the minimum total difficulty of playing the given right-hand passage with an optimal finger assignment using only entries of the given ergonomic tables. The optimal finger assignment may start and end with any finger.

Sample Input 1

```
11 11 11 10 10
1 2 1 1 1 1 1 1 1 2 2 2 2 3
1 3 1 1 1 1 1 1 1 1 1 2 2 3
1 4 2 2 1 1 1 1 1 1 1 1 1 2
1 5 3 3 2 2 1 1 1 1 1 1 1 1
2 3 1 1 1 1 2 2 3 3 3 3 3 3
2 4 2 2 1 1 1 1 2 3 3 3 3 3
2 5 3 3 2 2 1 1 1 1 1 2 2 2
3 1 2 2 3 3 4 4 4 4 4 4 4 4
3 4 1 1 2 2 3 3 3 3 3 3 3 3
3 5 3 3 1 1 1 1 3 3 3 3 3 3
4 5 1 1 1 1 3 3 3 3 3 3 3 3
1 2 1 1 1 1 1 1 1 1 2 2 3 0
1 3 1 1 1 1 1 1 1 1 1 1 2 0
1 4 2 2 2 1 1 1 1 1 1 1 1 0
1 5 3 3 3 2 2 2 1 1 1 1 1 0
2 3 1 1 1 2 2 3 3 3 3 3 3 0
2 4 2 1 1 1 1 2 2 2 3 3 3 0
2 5 3 2 2 2 2 1 1 1 2 2 3 0
3 1 4 4 4 4 4 4 4 4 4 4 4 0
3 4 1 1 1 3 3 3 3 3 3 3 3 0
3 5 3 2 2 2 2 2 3 3 3 3 3 0
4 5 2 2 2 2 3 3 3 3 3 3 3 0
1 2 3 2 2 1 1 2 2 2 3 3 3 0
1 3 3 2 2 1 1 1 2 2 2 2 3 0
1 4 3 3 3 1 1 1 1 1 2 2 2 0
1 5 3 3 3 2 2 2 1 1 1 1 1 0
2 3 1 1 1 2 2 3 3 3 3 3 3 0
2 4 2 1 1 1 1 2 3 3 3 3 3 0
2 5 3 2 2 1 1 1 1 1 1 1 2 0
3 1 2 3 3 4 4 4 4 4 4 4 4 0
3 4 1 1 1 3 3 3 3 3 3 3 3 0
3 5 2 1 1 1 1 1 2 2 3 3 3 0
4 5 1 1 1 2 2 3 3 3 3 3 3 0
1 2 0 2 2 2 2 0 3 3 3 3 0 3
1 3 0 2 2 2 2 0 2 2 2 2 0 2
1 4 0 3 2 2 2 0 2 1 1 1 0 2
1 5 0 3 3 3 3 0 2 1 1 1 0 1
2 3 0 1 1 1 2 0 3 3 3 3 0 3
2 4 0 2 1 1 1 0 2 3 3 3 0 3
2 5 0 3 2 2 1 0 1 1 1 2 0 2
3 4 0 1 1 1 2 0 3 3 3 3 0 3
3 5 0 3 1 1 2 0 3 3 3 3 0 3
4 5 0 2 2 2 3 0 3 3 3 3 0 3
33 34 39 42 38 41 39 42 46 51
```

Sample Output 1

```
10
```

This page is intentionally left blank.

Problem D
Froggie

You are recreating the classic 'Frogger' video game. You do not need to worry about sprites, music, or animation (that is left to the game's art team).

Each level consists of a road with multiple lanes, with cars traveling in both directions. Cars within each lane are evenly-spaced and move at the same speed and in the same direction. Lane directions alternate, with the top lane moving left to right.

'Froggie' starts below the bottom lane and she travels across the road from the *bottom* to the *top*. At each time step, Froggie hops one space in one of four directions: up, down, left, or right. The goal of the game is to get Froggie across the road and above the top lane without her getting hit by a car.

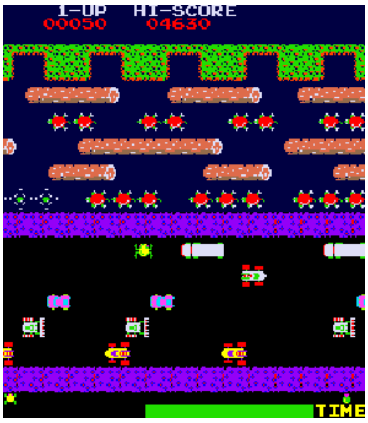


Photo from Wikipedia

Example

Consider the first sample input, where Froggie hops upward four times. The road has three lanes and a width of 7. Froggie starts below the road in the third cell. Cars in the top and bottom lanes are spaced at an interval of 3, and the middle lane has an interval of 2. The cars in the top two lanes move at a speed of 1, while those in the bottom lane move at a speed of 2. See Figure D.1.

At time 1 Froggie hops upward and the cars move. The cars in the top and middle lanes each move one space, and the cars in the bottom lane each move two spaces. See Figure D.2. After this first hop, Froggie is still safe. She now sits where a car *was*, and has avoided the path of the car as it travels. Notice that cars exit the simulated area as they travel off the grid, and also new cars also enter (at times that preserve each lane's interval).

At time 2, Froggie hops upward again as the cars continue to move. After hopping into the middle lane, Froggie *could* move left with this lane's traffic (traffic in this lane moves left at a speed of 1). See Figure D.3.

At time 3, Froggie hops upward a third time, reaching the top lane. See Figure D.4. Finally, at time 4, with a final upward hop, Froggie exits the road to safety. See Figure D.5.

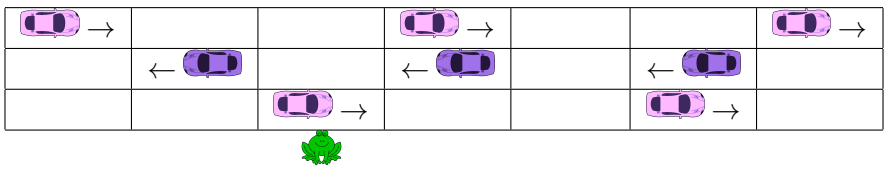


Figure D.1: Time 0. Cars and Froggie in their initial positions.

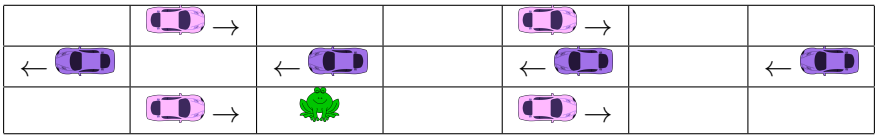


Figure D.2: Time 1. Froggie hopped up, the cars all moved. Some cars left, and some new cars appeared.

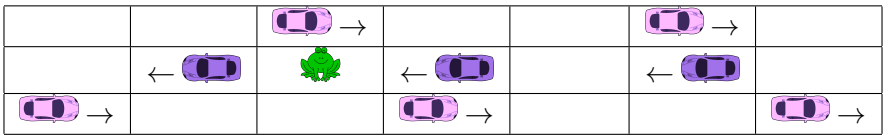


Figure D.3: Time 2. Froggie hopped up to the middle lane.

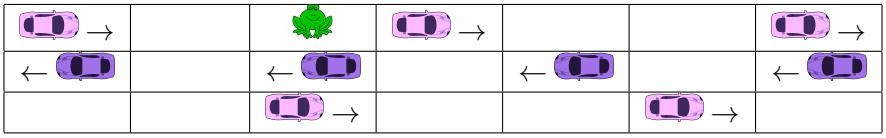


Figure D.4: Time 3. Froggie hopped up to the top lane.

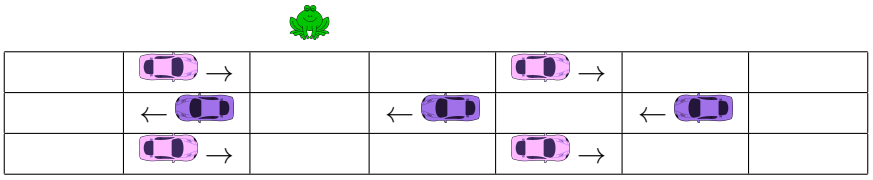


Figure D.5: Time 4. Froggie hops to safety!

Avoiding Cars

If Froggie enters the path of a moving car she is squished. Consider Figure D.6, which is a bit different than earlier figures in that it shows *one* car over *two* successive times. Let's say that at time t the car is in the leftmost cell and Froggie is not in this lane. At time $t + 1$, the car moves with a speed of 3 to the rightmost cell, and Froggie attempts to hop into the lane shown. Since the width of the lane is four, the only safe place to hop is the leftmost cell (where the car had just been). The other three cells would cause her to be squished by the car shown, ending the simulation.

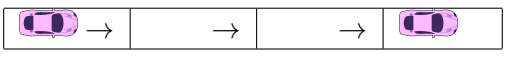


Figure D.6: A *single* car across two consecutive time steps, starting at the left and moving right.

Goal

Given a description of the road, car positions and speeds, and Froggie's starting positions and moves, determine her outcome after the simulation. There are two possible outcomes: safely exiting the top

lane or getting squished.

In order to better plan her travel, Froggie may move left or right before entering the road. Once Froggie has entered the road she may only exit through the top of the road. That is, Froggie's path never exits the left, right, or bottom lane boundaries.

Input

Each input describes one simulation. The first line of input contains two integers separated by a space: L and W. The number of lanes is given by L ($1 \leq L \leq 10$), while W ($3 \leq W \leq 20$) defines the width (in number of cells) of the grid.

This is followed by L lines each describing the car configuration for a lane (from top to bottom). Each line contains three integers: O, the starting offset of the first car; I, the interval between cars; and S, the speed of the cars. The bounds are $0 \leq O < I \leq W$ and $0 \leq S \leq W$. All offsets should be computed based on the direction of lane movement.

The final line of input starts with a single integer, P ($0 \leq P \leq W - 1$) followed by a space and then a string of M characters ($1 \leq M \leq L \cdot W$) from the set $\{\mathrm{U}, \mathrm{D}, \mathrm{L}, \mathrm{R}\}$. P defines the starting position of Froggie, starting from the left. The string of characters defines the sequence of moves (up, down, left, right) that Froggie makes during the simulation.

Output

If Froggie successfully crosses the road (exiting the road from the top lane), output "`safe`". If Froggie is hit by a car, or does not end above the road, output "`squish`".

Sample Input 1	Sample Output 1
3 7 0 3 1 1 2 1 2 3 2 2 UUUU	safe

Sample Input 2	Sample Output 2
3 6 0 3 1 1 2 1 2 3 2 2 U	squish

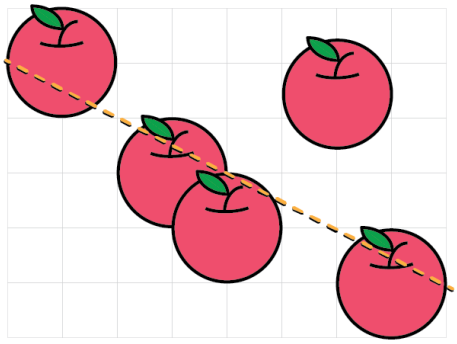
Sample Input 3	Sample Output 3
3 6 0 3 1 1 2 1 2 3 2 2 UR	squish

Sample Input 4	Sample Output 4
3 6 0 3 0 1 2 0 2 3 0 1 UUUU	safe

Sample Input 5	Sample Output 5
3 6 0 3 0 1 2 0 2 3 0 1 LRRLUUUU	safe

Problem E
Fruit Slicer

John, a student who is taking the game development course, recently developed a mobile game called *Fruit Slicer* for his coursework. In the game the player slices fruits that are throw into the air by swiping the touch screen. However the game is quite simple because John was not able to write code for the geometry required for a more complex version. In the game each slice is a straight line of infinite length, and all fruits have the same shape of a circle with unit radius. The figure shows a cool snapshot of John's game.



John introduces his game to his best friend Sean, who soon gets bored of playing the simple game. But as a teaching assistant of the algorithm course, Sean decides to turn the game into a homework assignment. He asks the students in the algorithms course to write a program that can compute the best slice at any given moment of the game. Given the locations of the fruits, the program should determine the maximum number of fruits that can be sliced with a single straight-line swipe.

As a student in Sean's class, you are now the one who is facing this challenge.

Input

The first line has a single integer n ($1 \leq n \leq 100$). The next n lines each have two real numbers giving the x and y coordinates of a fruit. All coordinates have an absolute value no larger than 10^4 and are given with exactly two digits after the decimal point. Fruits may overlap.

Output

Output the maximum number of fruits that can be sliced with one straight-line swipe. A swipe slices a fruit if the line intersects the inner part or the boundary of the fruit.

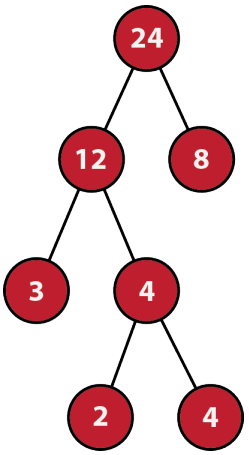
Sample Input 1	Sample Output 1
5 1.00 5.00 3.00 3.00 4.00 2.00 6.00 4.50 7.00 1.00	4

Sample Input 2	Sample Output 2
3 -1.50 -1.00 1.50 -1.00 0.00 1.00	3

Sample Input 3	Sample Output 3
2 1.00 1.00 1.00 1.00	2

Problem F
LCM Tree

An LCM tree is a binary tree in which each node has a positive integer value, and either zero or two children. If two nodes x and y are the children of node z, then the *Least Common Multiple (LCM)* of the values of node x and node y must equal the value of node z.



You are given n nodes with positive integer values to be arranged into an LCM tree. In how many ways (modulo $10^9 + 7$) can you do that? Two ways are considered different if there are two nodes x and y so that x is a child of y in one way but not in the other way.

The illustration shows one of the two ways for the first sample case. The other way can be obtained by swapping the two nodes with value 4. Note that swapping the two leaves with values 2 and 4 does not give a different way.

Input

The first line has an odd integer n ($1 \leq n \leq 25$). The second line has n positive integers no larger than 10^9, giving the values of the nodes.

Output

Output the number of ways to arrange the given nodes into an LCM tree, modulo $10^9 + 7$.

Sample Input 1	Sample Output 1
7 2 3 4 4 8 12 24	2

Sample Input 2	Sample Output 2
3 7 7 7	3

Sample Input 3	Sample Output 3
5 1 2 3 2 1	0

Sample Input 4	Sample Output 4
13 1 1 1 1 1 1 1 1 1 1 1 1 1	843230316

This page is intentionally left blank.

Problem G
Left and Right

Image from Pixabay

With modern technology advancement, it is now possible to deliver mail with a robot! There is a neighborhood on a long horizontal road, on which there are n houses numbered 1 to n from left to right. Every day a mail delivery robot receives a pile of letters with exactly one letter for each house. Due to mechanical restrictions, the robot cannot sort the letters. It always checks the letter on top of the pile, visits the house that should receive that letter and delivers it. The robot repeats this procedure until all the letters are delivered. As a result, each of the n houses is visited by the robot exactly once during the mail delivery of a single day.

The mail delivery robot has a tracking device that records its delivery route. One day the device was broken, and the exact route was lost. However, the technical team managed to recover the *moving directions* of the robot from the broken device, which are represented as a string consisting of $n-1$ letters. The i-th letter of the string is '`L`' (or '`R`') if the $(i+1)$-th house visited by the robot is on the left (or right) of the i-th house visited. For example, if $n = 4$ and the robot visited the houses in the order of $2, 4, 3, 1$, its moving directions would be "`RLL`".

With the moving directions, it may be possible to determine the order in which the robot visited the houses. The technical team has asked you to write a program to do that. There can be multiple orders producing the same moving directions, among which you should find the lexicographically earliest order.

Input

The input has a single integer n ($2 \le n \le 2 \cdot 10^5$) on the first line. The second line has a string of length $n-1$ consisting of letters '`L`' and '`R`' giving the moving directions of the robot.

Output

Output the lexicographically earliest order in which the robot may have visited the houses and delivered the letters according to the moving directions.

Sample Input 1	Sample Output 1
3 LR	2 1 3

Sample Input 2	Sample Output 2
6 RLLRL	1 4 3 2 6 5

Sample Input 3	Sample Output 3
6 RRRLL	1 2 3 6 5 4

Problem H
Longest Life

Illustration by William Ely Hill

Everyone wants to live as long a life as possible. As time progresses, technology progresses. Various anti-aging pills get introduced to the market at various times which allow a person to age more slowly than normal. In particular, an x-y pill, when taken regularly, ages your body only y seconds over the course of x seconds. So, if you took a 100-87 pill regularly, then over the next 100 seconds, your body would only age 87 seconds. You can only take one of these pills at a time (or none at all). The only downside to using one of these pills is that due to the change in regimen, if you switch to a pill, it automatically ages you c seconds. The value of c is the same for all pills.

Any time you switch to an x-y pill, you can take it for any number of seconds, but you can only switch to a pill at or after the time it first becomes available on the market. For the purposes of this problem assume that your life starts at time $t = 0$ seconds and that without the aid of any pills, you would live to be n seconds old.

Given information about each different pill introduced into the market (the time it is introduced, in seconds, and its corresponding x and y values as previously described) and c, the number of seconds you automatically age when switching pills (or switching to a pill from no pill at all), determine the longest you can live, over all possible schedule of pills you could take.

Input

The first line of input consists of three positive integers, n ($n \leq 3 \cdot 10^9$), representing the number of seconds you would live without taking any pills, p ($p \leq 10^5$), the number of pills that become available on the market, and c ($c \leq 10^5$), the time in seconds you age as soon as you switch to a different pill. p lines follow with the i^{th} line containing three space separated integers: t_i ($1 \leq t_i \leq 10^{12}$), x_i and y_i ($1 \leq y_i < x_i \leq 10^4$), representing the time the i^{th} pill gets introduced to the market, and the corresponding x and y values for it. In addition, for all i, $1 \leq i \leq n-1$, it is guaranteed that $t_{i+1} - t_i > c$.

Output

Output a single real number, representing the maximum number of seconds that you could live, if you take the appropriate pills. Your answer should be correct within a relative or absolute error of 10^{-6}.

Sample Input 1	Sample Output 1
100 3 10 15 99 98 40 3 2 90 10 9	115.000000000

Sample Input 2	Sample Output 2
10000 4 100 1000 1001 1000 1994 10 9 2994 100 89 3300 1000 1	6633900.000000000

Problem I
Monitoring Ski Paths

Photo by Darryns

Fresh powder on a sunny day: it is a great time to ski! Hardcore skiers flock to a large mountain in the Rockies to enjoy these perfect conditions. The only way up the mountain is by helicopter; skiers jump out and ski down the mountain.

This process sounds a bit chaotic, so some regulations are in place. Skiers can only enter or exit the mountain at a set of designated locations called *junctions*. Once on the mountain, they are only allowed to travel along designated *ski paths*, each of which starts at one junction and ends at another junction lower on the mountain. Multiple ski paths might start at the same junction, but no two ski paths end at the same junction to avoid collisions between skiers.

Finally, each skier must register a ski plan a day in advance with the helicopter service. The ski plan specifies the junction they fly up to and the junction lower on the mountain where they get picked up. If a skier shows up to the mountain, they must follow their plan; but some skiers get sick and do not show up at all.

Your job is to look through the ski plans and set up monitors at some of the junctions to count how many skiers actually show up. To keep operating costs as low as possible, you should determine the minimum number of junctions that need to be monitored so that each skier passes through *at least one* monitored junction.

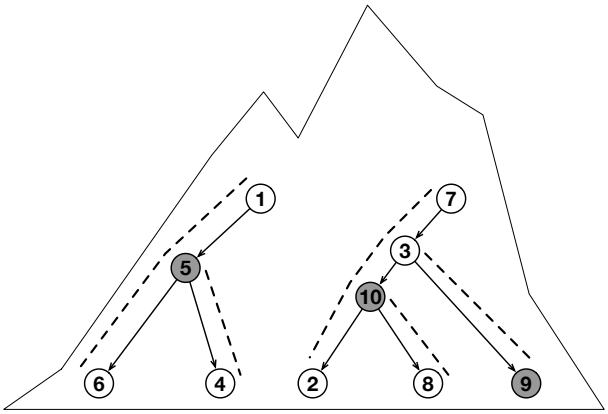


Figure I.1: Illustration of the first sample input.

Figure I.1 shows the first sample input. The dashed lines indicate the five different plans that were registered by skiers. By monitoring junctions 5, 9, and 10, you can ensure that all plans include at least one monitored junction. Monitoring fewer functions would miss some skiers.

Input

The first line of input has three integers n, k, and m, where n ($2 \leq n \leq 250\,000$) is the number of junctions, k ($1 \leq k < n$) is the number of ski paths, and m ($1 \leq m \leq 250\,000$) is the number of routes.

Then k lines follow, each containing two integers $1 \leq u, v \leq n$ indicating that there is a ski path that starts at junction u and ends at junction v.

Then m lines follow, each containing two distinct integers $1 \leq s, t \leq n$. Each line indicates that a skier plans to land at junction s and ski down the mountain to junction t. It is guaranteed it is possible to reach junction t from junction s by following ski paths and that junction t is at the base of the mountain (i.e. no ski paths start at t). No (s, t) pair appears more than once.

Output

Output the minimum number of junctions that need to be monitored so each ski plan includes at least one monitored junction.

Sample Input 1	Sample Output 1
10 8 5 1 5 5 6 5 4 7 3 3 10 3 9 10 2 10 8 1 6 5 4 7 2 3 9 10 8	3

Sample Input 2	Sample Output 2
6 5 4 1 2 4 6 2 3 4 5 2 4 1 6 2 6 1 3 2 5	1

Problem J
Peg Game for Two

Photo by ceratosaurrr.

Jacquez and Alia's parents take them on many road trips. To keep themselves occupied, they play a triangular peg game, meant for one player. In the original game, one player has an equilateral triangle containing 15 holes (five holes per side of the triangle). Initially one hole is empty and the remaining 14 are filled with pegs. The player moves by picking up a peg to "jump" it over another peg, landing on an empty hole. Jumps must be in straight lines (horizontally or diagonally). The peg that is jumped over is removed. The goal of this game is to leave just one peg on the board. Figure J.1 shows the result of a jump in this game.

```
    X                   X
   X X                 X O
  X X X      -->      X O X
 X O X X             X X X X
X X X X X           X X X X X
```

Figure J.1: One possible move in the original game. On the left is an initial board, where `X` represents a peg and `O` represents an empty hole. The jump moves the right peg in the second (from top) row diagonally down and to the left, over the middle peg of the third row, which is removed.

Ultimately, Jacquez and Alia have gotten bored of playing this game individually. They decide to invent a version that they could play together. In the new version, each peg has a positive point value, and they alternate turns. During a player's turn, they must make a jump if one is available. The score for a jump is the product of the two pegs involved in the jump.

The total score of a player is the sum of the scores of their jumps. The game ends when a player has no possible jumps to make. Each player's goal is to maximize their own total score minus their opponent's total score (at the end of the game). For example, Jacquez would prefer to win with a score of 100 to Alia's score of 60 (with a differential of 40), than to win with a score of 1 000 to Alia's score of 998 (with a smaller differential of only 2). Similarly, Alia wants to win by as much as possible over Jacquez.

For this version of the game, we can display the game state by replacing each `X` shown above with a corresponding value for that peg, and using a value of 0 (zero) for each of the open holes. Figure J.2 shows an example move.

Jacquez has had some trouble winning. Write a program to calculate the best he can do, assuming that he starts and both players play optimally.

Input

The input consists of five lines, describing the initial state of the board. The i^{th} $(1 \leq i \leq 5)$ line contains i space separated integers, representing the pegs and holes on the i^{th} row. Each of these integers is

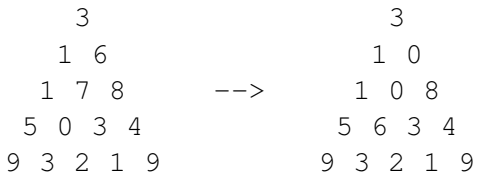


Figure J.2: One possible move in the new game, jumping the peg with value 6 over the peg with value 7. This move is worth $6 \cdot 7 = 42$ for Jacquez (since he moves first). The values are from the first sample input.

between 0 and 100, inclusive. A value of 0 represents a hole while the rest of the values represent pegs. It is possible for two different pegs to have the same value. Of the 15 input values it is guaranteed that exactly one is 0, thus all input boards start with exactly one hole.

Output

Output the value of Jacquez's score minus Alia's score at the end of the game, assuming Jacquez starts and both players play optimally.

Sample Input 1	Sample Output 1
3 1 6 1 7 8 5 0 3 4 9 3 2 1 9	21

Sample Input 2	Sample Output 2
1 2 3 4 5 6 7 8 9 10 11 12 0 13 14	19

Sample Input 3	Sample Output 3
100 1 17 99 3 4 0 76 33 42 12 13 14 15 16	2148

Problem K
Run-Length Encoding, Run!

Photo by secretlondon123

Forrest lives in a prehistoric era of "dial-up Internet." Unlike the fast streaming of today's broadband era, dial-up connections are only capable of transmitting small amounts of text data at reasonable speeds. Forrest has noticed that his communications typically include repeated characters, and has designed a simple compression scheme based on repeated information. Text data is encoded for transmission, possibly resulting in a much shorter data string, and decoded after transmission to reveal the original data.

The compression scheme is rather simple. When encoding a text string, repeated consecutive characters are replaced by a single instance of that character and the number of occurrences of that character (the character's *run length*). Decoding the encoded string results in the original string by repeating each character the number of times encoded by the run length. Forrest calls this encoding scheme *run-length encoding*. (We don't think he was actually the first person to invent it, but we haven't mentioned that to him.)

For example, the string `HHHeelllo` is encoded as `H3e2l3o1`. Decoding `H3e2l3o1` results in the original string. Forrest has hired you to write an implementation for his run-length encoding algorithm.

Input

Input consists of a single line of text. The line starts with a single letter: `E` for encode or `D` for decode. This letter is followed by a single space and then a message. The message consists of 1 to 100 characters.

Each string to *encode* contains only upper- and lowercase English letters, underscores, periods, and exclamation points. No consecutive sequence of characters exceeds 9 repetitions.

Each string to *decode* has even length. Its characters alternate between the same characters as strings to encode and a single digit between 1 and 9, indicating the run length for the preceding character.

Output

On an input of `E` output the run-length encoding of the provided message. On an input of `D` output the original string corresponding to the given run-length encoding.

Sample Input 1	Sample Output 1
`E HHHeellloWooorrrrlld!!`	`H3e2l3o1W1o3r4l2d1!2`

Sample Input 2	Sample Output 2
`D H3e2l3o1W1o3r4l2d1!2`	`HHHeellloWooorrrrlld!!`

This page is intentionally left blank.

Problem L
Superdoku

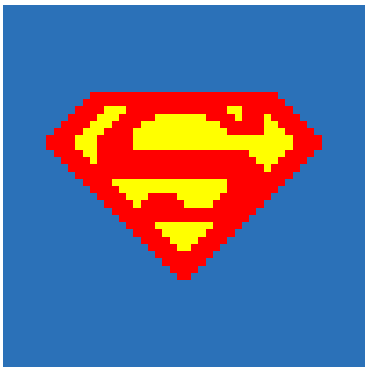


Illustration by jukeboxhero.

Alice and Bob are big fans of math. In particular, they are very excited about playing games that are related to numbers. Whenever they see a puzzle like Sudoku, they cannot stop themselves from solving it. The objective of Sudoku is to fill a 9×9 grid with digits so that each column, each row, and each of the nine (3×3) subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

After many years of solving Sudoku problems, Alice and Bob are tired of Sudoku. They have been trying to develop a harder variation of Sudoku, which they are calling Superdoku. In Superdoku, the grid is bigger – $n \times n$ instead of just 9×9. However, the "block" constraints are impossible to formulate when there are no further constraints on n. Therefore, there are no block constraints in Superdoku. Instead, the goal is simply to make sure that each column and each row in the grid contains all of the integers from 1 to n. After playing for a while in the standard way (where any of the grid cells may have previously been filled in), they decide that the game is too difficult and they want to simplify it. Therefore, they decide to make the initial grid further constrained. They constrain the board by filling in the first k rows completely.

Alice and Bob both believe that Superdoku is solvable. However, since n could be very big, it may still take a long time to figure out a solution. They don't want to spend too much time on this single game, so they are asking for your help!

Input

The input consists of a single test case. The first line lists two space-separated integers $1 \leq n \leq 100$ and $0 \leq k \leq n$, denoting the size of the grid $(n \times n)$ and the number of rows k that are already filled in. Each of the following k lines contains n space-separated integers, denoting the first k given rows. All integers in these k lines are between 1 and n.

Output

Output either "`yes`" or "`no`" on the first line, indicating if there is a solution. If there is no solution, do not output anything more. If there is a solution, output n more lines, each containing n space-separated integers, representing a solution. If there are multiple solutions, output any one of them.

Sample Input 1	Sample Output 1
4 2 1 2 3 4 2 3 4 1	yes 1 2 3 4 2 3 4 1 3 4 1 2 4 1 2 3

Sample Input 2	Sample Output 2
4 2 1 2 3 4 2 2 2 2	no

Problem M
Triangular Clouds

Photo by SketchUp

Garry is looking at the sky. Such a beautiful day! He notices that the clouds are particularly beautiful today, and wishes to record the current state of the sky. He has no camera, so he begins writing down coordinate points. Fortunately for Garry, the current cloud cover can be represented as the union of non-intersecting, non-degenerate triangles where each vertex is at a coordinate point on the xy-plane. Two triangles are considered non-intersecting if their intersection has area 0.

The next day, Garry's friend Jerry goes to look at the sky. Jerry also wishes to record the current state of the sky. He follows the same protocol as Garry, and writes down the cloud cover as a set of non-intersecting triangles.

Garry and Jerry want to determine if they saw the same cloud cover. Unfortunately, there are multiple possible ways to represent the same cloud cover! Given Garry and Jerry's notes, did they see the same cloud cover in the sky?

Input

The first line of input contains the integer n, $(0 \leq n \leq 100\,000)$, the number of triangles Garry wrote down. Each of the next n lines contains 6 space separated integers, x_1, y_1, x_2, y_2, x_3, and y_3. These are Garry's triangles. The next line contains the integer m, $(0 \leq m \leq 100\,000)$, the number of triangles Jerry wrote down. Each of the next m lines contains 6 space separated integers, x_1, y_1, x_2, y_2, x_3, and y_3. These are Jerry's triangles. The absolute value of the x and y coordinates are at most 10^9. (That's as far as Garry and Jerry can see.)

Output

Print "`yes`" if Garry and Jerry saw the same cloud cover, or "`no`" if they did not.

Sample Input 1	Sample Output 1
1 10000 0 10000 10000 0 10000 3 10000 0 10000 10000 5000 5000 5000 5000 10000 10000 0 10000 0 0 0 1 1 0	no

Sample Input 2	Sample Output 2
2 9996 0 9997 0 0 1 9999 0 10000 0 0 1 2 9997 0 9998 0 0 1 9998 0 9999 0 0 1	no

Sample Input 3	Sample Output 3
1 2 0 2 4 4 2 2 2 0 2 2 4 2 2 2 4 2 2 4	yes