Problem A. Campus

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

There are lots of companies in the Bay Area. Some of them are so big that they need a giant campus for their engineers. But most of the buildings are already occupied, so it is hard to find vacant buildings located next to each other.

Former participants of the first Izhevsk training camp founded a new company in the Bay Area. Now they need to rent three buildings for their engineers.

Currently there are n buildings out for rent in the Bay Area. You can assume that each building is a point on a plane defined by its coordinates. There are no two buildings with equal coordinates.

The founders don't want their buildings to be located on one line. For the efficient work there should be a good network connection between the buildings. Each building has its own connection value. You need to maximize the sum of the connection values for the chosen buildings to achieve fast and stable network.

Do you want to join the company? This is your first interview problem.

Input

On the first line you are given a single integer n ($3 \le n \le 10^5$) — number of buildings in the Bay Area. On the next n lines there are three integers x_i, y_i, s_i ($-10^5 \le x, y \le 10^5, 0 \le s_i \le 10^5$) — coordinates and the connection value for i-th building.

Output

On the first line output the maximum sum of the connection values of the chosen three buildings which are not located on one line. On the second line output the indices of those buildings in any order. The buildings are numerated from 1 to n in the same order as they are given in the input. It is guarantee that there exist three points which are not located on the same line.

Examples

standard input	standard output
3 0 0 1 0 1 2 1 0 3	6 3 2 1
5 0 0 1 0 1 2 0 2 3 0 4 4 1 0 5	12 5 4 3

Problem B. Black hole (Division 1 only!)

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

In 3141 the black hole problem become very big for the citizens of San Francisco. San Francisco is located not on the Earth, but in space. It is a network of orbit stations which are connected between each other with tunnels. The tunnels could not be broken or become disconnected. The tunnels can freely pass through each other.

In this problem you can assume that San Francisco is a closed polyline in *3D* space without self-intersection and touching.

The black hole can trap the orbit stations and tunnels. To protect San Francisco from that it was decided to build the special devices in the open space. This devices generate the lines. Nothing could intersect these lines, neither orbit station, nor tunnel. The devices were placed in the space. But nobody sure that the devices were located correctly.

The devices are located correctly if there is no black hole that could trap San Francisco — the lines will not let tunnels pass through them. A black hole could appear in any point of the space except the orbit station or a tunnel. Also a black hole cannot appear on any line. The tunnels cannot be disconnected. If the devices are located incorrectly then San Francisco can be moved to a black hole completely and the lines can not help.

Input

On the first line of the input file you are given a single integer n — the number of vertices of the polyline ($3 \le n \le 1000$). On the next n lines you are given the description of the vertices of the polyline — integers x, y, z ($-10^4 \le x, y, z \le 10^4$). On the next line you are given one single integer m ($1 \le m \le 1000$) — number of lines. All lines are parallel to the OZ axe. On the next m lines you are given lines description — integer coordinates x, y ($-10^4 \le x, y \le 10^4$). It is guaranteed that all the given lines do not have common points with polygon.

Output

Output one single word "`Yes`", if the polyline could be adsorbed to some point not located on the devices without breaking its edges, otherwise output "`No`".

Examples

standard input	standard output
4 0 0 0 2 0 0 2 2 0 0 2 0 1 1 1	No
4 0 0 0 2 0 0 2 2 0 0 2 0 1 3 3	Yes

Problem C. Cubes (Division 1 only!)

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

In San Francisco there are lots of amazing stores for kids: candies, books and, of course, toys! These stores are always crowded by children who want a toy. But parents are not so fond of buying something just because their kid is crying, they buy toys for Christmas. After one Christmas little Billy found an amazing set of color cubes in the sock.

Billy played with his cubes for a long time. His favorite game was building a wall. He has built a lot of different walls with different lengths and colors. Billy's mother was so proud of her son that she posted photos of all the walls from all directions to her "Instakilogram". Once Billy's mother received message that some of her photos had been gone.

Billy's mother found only two photos on her account that were made from left and from right direction to the wall. To be more specific: if we look at the wall from the front direction we can see the construction with b towers standing next to each other forming the wall. One tower is a set of cubes $1 \times 1 \times h_i$, where h_i is the height of the i-th tower. The height of the tower could be zero. If we look at the wall from one of its side (left or right direction) we can see one tower only (but each cube of the tower could really belong to different towers).

Billy's mother also found the comment under the picture with the quick description: "Billy has built a wall of length no more than b, used cubes of no more than k different colors, the highest tower had height of exactly h". Now she is wondering how many walls could be built by Billy. The number of walls could be huge, so she wants to find the answer modulo 1000000007 ($10^9 + 7$).

Input

The first line of the input contains three integers b, k and h ($1 \le b, k, h \le 10^5$) — the length of the wall, the number of different colors that could be used in the wall (not necessary all of them should be used), and the height of the tower of cubes on the left and right direction picture.

The second line contains the description of the left direction tower: h integers c_i ($1 \le c_i \le k$), where c_i is the color of the i-th cube counting from bottom to top. The third line contains the description of the right direction picture in the same format.

Output

Print the number of walls that could match to these pictures modulo 1000000007 ($10^9 + 7$).

Examples

standard input	standard output
3 3 3 1 2 3 3 2 3	132
2 3 5 1 2 3 2 1 3 2 1 2 1	5
1 1 1 1 1	1

Note

The explanation of the second sample test:

```
11    1    1   1    1
22   22    2   22   2
31   31   31   31   31
22   22   22   22   22
13   13   13   13   13
```

Problem D. Garbage problem (Division 1 only!)

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

The inhabitants of San Francisco have realized that there is a huge garbage problem and someone should immediately start thinking about the solution. But first let's understand what is going on. Scientists concluded that there were several dirty regions in the city and these regions were growing every second. Now the problem that the scientists need to solve is to find out when the garbage areas would separate the city into non-connected components.

In this problem we can assume that the city is a polygon without holes, self intersections and touching. In the beginning each dirty region is a circle with zero radius. All dirty regions are growing with the speed 1, that is at time t radius of each region is t.

You need to figure out the first moment of time when there would be more than one clean region in the city and they would not be connected to each other. And you need to find out the number of components as well.

Input

The first line of the input contains integer n ($3 \le n \le 100$) — the number of vertices of the polygon that represents the city. Next n lines contain the points of the polygon in the traverse order.

Next line contains integer k ($1 \le k \le 20$) — the number of dirty regions. Next k lines represent the centers of dirty regions, one point on each line. Center of each dirty region is either inside the polygon or on the border of the polygon.

All coordinates are integers in range $[-10^5, 10^5]$.

Output

On the first line of the output write "`Never`" if the event never happens, or "`Yes`" if it is possible.

For the "`Yes`" output, second line should contain first moment of time when the city becomes disconnected. Absolute or relative error of the answer must not exceed 10^{-5}. The third line should contain the number of disconnected components.

Examples

standard input	standard output
4 0 0 0 5 9 5 8 0 1 2 4	Yes 2.0 2
4 0 0 8 0 8 4 0 4 2 4 1 4 3	Yes 1.0 2
3 0 0 7 2 2 4 1 2 4	Yes 3.2966535347685366 2

Problem E. Interview

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

Interviews are a lot of fun. Imagine, if you will, that you are going to an interview to solve a problem. You could have just solved it at home, but indulge me for a moment and imagine instead that you're at an interview.

This interview question was prepared very carefully. Every interviewer wants to ask the hardest question, and one of the best topics to ask about is trees.

You're given a question by an interviewer. It's a hard problem that he himself couldn't solve. Help him, please.

The problem might sound simple: calculate the number of labeled trees with n vertices such that for each vertex the number of neighbors that are leaves is no more than k. The answer could be large, so output it modulo $10^9 + 7$.

A labeled tree is a tree the vertices of which are assigned unique numbers from 1 to n.

Input

On the single line you are given two integers: n, k ($2 \le n \le 100, 1 \le k \le 100$) — number of vertices of the tree and the limit on the number of neighbor leaves.

Output

Output the answer modulo 1000000007 ($10^9 + 7$).

Examples

standard input	standard output
3 3	3
3 1	0

Note

Examples of the labeled trees with two and three vertices:

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

Problem F. Reverse engineering

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

As you know, San Francisco is the center of Software Engineering. Most difficult problems are solved there. One of the companies found the problem which they still cannot solve. Help them, please.

The description is simple: "Black box". There is some program that gets something as input and produces something to the output. You are given this black box (in the supplementary problem **Q**) and some sample tests just to start with.

The only requirement that you know is that the "Black box" accepts only strings no longer than 1000 characters. You need to repeat the logic of the "Black box".

Input

A string with the length no more than 1000 and valid ASCII code characters from 32 to 127.

Output

Your output for each test cases should be the same as the "Black box".

Examples

standard input	standard output
`int foo;`	`Declare foo as int`
`double *arr[23];`	`Declare arr as array 23 of pointer to double`
`double (*a)(int, double (*)()))(int);`	`Incorrect input`

Note

Start with simple tests to find out what is valid and what is not.

Problem G. Cover WiFi

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

San Francisco is amazing city! It is located near the ocean and you can easily go for a walk on the beach. All important locations are connected with bridges. And as you can assume, there are lots of cars on the bridges during the business hours. The traffic is awful.

Government of California decided that it is not perfect that people in their state are upset about the traffic. So the idea came suddenly: WiFi! If people could serf in the web while stuck in the traffic it would make them happy! But nobody likes spending more money than it's needed. So Government has decided to set up only one hotspot and now they need to determine the location.

You could imagine that all the problematic locations are just points on the plane. Government has a plan to set up the hotspot in the way to cover not less than k problematic points. Of course, they want to use the least powerful hotspot as possible (power of hotspot would be measured by the radius of coverage). So your task is to find the minimal possible radius to cover at least k problematic locations, and the location of the hotspot as well.

Input

The first line contains two integers n and k ($2 \le n \le 10000, 2 \le k \le min(50, n)$) — the number of problematic locations and minimum number of locations you need to cover. Next n lines contain coordinates of the locations — integers x, y ($-10000 \le x, y \le 10000$). There are no two equal locations.

Output

On the first line output the minimum radius of the hotspot with relative or absolute error no more than 10^{-5}. Next line should contain the coordinates of the location with the same acceptable error.

Examples

standard input	standard output
5 4 0 0 1 1 0 1 1 0 10 10	0.7071067813839326 0.5000000000000001 0.4999999997208554
3 3 0 0 1 1 0 1	0.7071067813839326 0.49998818593030947 0.5000118140696904

Problem H. Poker again

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

After a contest, a group of software engineers decided to play poker. Soon the number of chips in front of each player became different. Someone had a big pile of chips, another just rolled the last chip in his hand. At the end of each game, players showed their cards and winner was determined. The problem is to help players to calculate the result of the game and determine who wins and who loses.

To simplify the problem let's look at only the last round of bets (in *Texas Holdem* there are four rounds of bets before the players open the cards and determine the winner). In general the game looks like this: players are dealt two cards each from 52 cards deck. Each player knows his own cards, but not the cards of the opponents. Then there are four rounds of bets. After each round *dealer* (we define him later) reveals some cards. After the first round *dealer* reveals three cards, after the second round one more card, after the third one one more card. So there are *five* cards revealed on the table on the last (fourth) round of bets.

The deck consists of 52 cards: 13 ranks and 4 suits for each rank. Suits are: *C, D, S, H*. Ranks from the smallest: *2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A*.

The card combination for the player is formed from his own cards plus the cards that are revealed on the table — total *seven* cards. *From these seven cards five cards which create the best combination are chosen.* These are all combinations starting from the best one:

- *"Straight flush"*: Look at *"Straight"* and *"Flush"*. If both *"Straight"* and *"Flush"* are taking place then the combination is called *"Straight flush"*. If multiple players have *"Straight flush"*, the winner is the player whose high rank card for the *"Straight"* is higher. Example: *5H 6H 7H 8H 9H*.
- *"4 of a kind"*: Four cards of one rank. If multiple players have *"4 of a kind"*, first, we need to compare the rank of those four cards, then in the case of a tie we compare the rank of the last card. Example: *5H 5C 5D 5S 9H*.
- *"Full house"*: Among the five cards there are three cards with the same rank, and the other two cards also have the same rank. In the case of a tie, we need to compare the rank of the three cards, and if it is still a tie then the rank of the other two. Example: *5H 5S 5D 6H 6D*.
- *"Flush"*: All five cards have one suit. In the case of a tie we compare the highest rank among all five cards, if it is still a tie then we compare the next one and so on. Example: *5H 6H 7H 8H AH*.
- *"Straight"*: ranks of the five cards are consecutive. If multiple players have a *"Straight"*, the winner is the player whose high rank card for the *"Straight"* is the highest. Ace could open the *"Straight"* or could be its end. Examples: *5H 6H 7H 8H 9D*, *AH 2D 3C 4S 5C*, *TH JC QS KS AH*. Note that the king cannot open *Straight*.
- *"3 of a kind"*: Among the five cards there are three cards with the same rank. If several players have *"3 of a kind"*, we compare the rank of those three cards, then in the case of a tie the highest rank of the last two cards and in the case of a tie the rank of the last card. Example: *5H 5C 5D KH AD*.

- *"2 pairs"*: Among the five cards there are two cards with the same rank, and from the rest three cards there are two cards with the same rank. In the case of a tie we compare the highest rank for the pairs, then if it is still a tie we compare the next rank for second pairs, and then the remaining card. Example: *5H 5C 6D 6H AD*.
- *"Pair"*: Among the five cards there are two cards with the same rank. In the case of a tie we compare the rank for the pair, then if it is still a tie we compare rest cards from the highest rank. Example: *5H 5C 2D KH AD*.
- *"High card"*: We compare the highest rank among all five cards, if it is still a tie then we compare the next one and so on. Example: *5H 4C JD TH AD*.

Let's describe the last round of bets. The player cannot bet the number of chips that is less than some fixed minimum value (*Blind*). There is a bank with all chips from the previous rounds bets. We can assume that the bank size is equal to the *Blind* size multipled by number of players (i.e. each player bet exactly one *Blind* in all previous rounds). One of the players dealt the cards. This player is called *dealer*. The next player to the left of the dealer (in clockwise order) bets first. During bets players says their actions turn by turn. Action of one player could increase the number of chips in the bank.

To describe all possible actions of the players, let's introduce the term "current stack level". "Current stack level" is maximum number of chips which a player put in the bank during current round of bets. In the beginning of the round "current stack level" equals to zero, and then it could increase. To continue the game, player should put the number of chips to his personal stack at this round that would be the same or more than current stack level is, or put all of the remaining chips if he doesn't have enough to match the current stack level.

Here is the list of possible players actions:

- *Fold*. Put nothing to the bank and leave the game. All chips which player has already put earlier remain in the bank (i.e. he loses them).
- *Check* or *Call*. Put minimum number of chips to the bank to match current stack level. Check means that he puts no chips, and *Call* means that he puts non-zero number of chips to the stack.
- *Raise X*. Player puts the number of chips that current stack level in the way he would increase the stack level by the value X. X must be greater or equal than the *Blind* size.
- *All in*. Put all remaining chips into the stack. This action may or may not increase the current stack level, depending on how many chips player has. This action could be done irrespective of stack level, given that player is still in game. Also any action which results putting all remaining chips into the game is considered as *All in*, even if it's *Call* or *Raise* at the same time.

After a player says *All in* or *Fold* he doesn't participate in the round of bets any more (he doesn't say anything when it is his turn). The bets round ends when everybody who participates in the game agreed with the bet. In the other words the turn returned back to the player who raised last. In this case he says nothing and the bets round end.

Now let's describe how the players open the cards. That happens after the bets round. If there is only one player left in the game he doesn't open his cards. Otherwise the first player who opens the cards is the one who raised last (*Raise* or *All in* with increase of current stack level). If nobody raised the bet then the first player who opens his cards is the second to the left from the *dealer* (in clockwise order). The player will open his cards if and only if he has any chance to win any number of chips. If the player knows that he's lost he doesn't open the cards. Not opening the cards equals to *Fold*.

Then the rank of each player's combination is calculated and profits are determined. Player can't win more chips than he actually bet. If several players bet different number of chips, bank is split into several banks. For example, let's assume that there is a game of three players and the number of chips for the corresponding player is 10, 100 and 1000. If the first player says *All In* and both the second and the third

players calls (i.e. each put 10 into the bank) then the bank is 30 chips. But if the second player says *Raise* 50 and third one calls (i.e. second and third players put 60 chips each to the bank) then the bank splits to several banks: first one is the bank with the 30 chips. All the players (first, second and third) are taking part in the game for this bank. In the second bank there are 100 chips and only two players are competing for this bank: second and third players. In each game the player who has the best combination takes all chips from the bank. In case of a tie, bank is split equally between all winners. The remainder after the division goes to the dealer for the good deal as the tips (even if dealer doesn't take part in this game).

In this problem you are given a full picture of the table: all card of all players, their initial number of chips and all their actions during the last bets round. You should ensure that all their actions were correct according to the rules described above. Also determine the players who will show their cards and number of chips each player gains (or loses) after the game.

Input

On the first line of the input file you are given three integers: number of players — n ($3 \le n \le 8$), *Blind* size — b ($20 \le b \le 100$) and the player who is a *dealer* — id ($1 \le id \le n$). On the next line there are n integers separated by space — number of chips the corresponding player currently has — c_i ($1 \le c_i \le 20000$). Then you are given n lines with the description of the cards each player has on hands. Each description consists of two strings, each string consists of two characters. First on is the rank of the card, second is the suit of the card. Suits are capital English letters: C, D, S, H. Ranks are: $2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A$ (all letters are capital).

On the next line in the same format you are given five cards that were revealed on the table. **It is guaranteed that all the cards information is correct**.

Then you are given a game log. On the first line there is a single integer m — number of the lines in the log ($1 \le m \le 50$). Then on the next m lines there are players' claims. One line has a single player claim. See the example to verify the format. Claims could be: *Fold*, *Check*, *Call*, *Raise* X (where X is a positive number of chips, the player raised the bet $X \le 20000$) and *All in*. **It is guaranteed that there are no typos in the claims and all the claims formats are correct**.

Output

If the game log was incorrect, output "Incorrect game log" without the quotas. If the log was correct then output on the first line one single word "Correct". On the next line output n integers, separated by space. Each integer is a number of chips the corresponding player would have after the game. On the next line output a single string consisting of zeros and ones with the length n, where each character is one if the player opened his cards, zero otherwise.

Examples

standard input
3 20 2 50 100 120 AH AC KH KD JD JS JH KS KC AD AS 6 Player 3: Raise 20 Player 1: All in Player 2: Call Player 3: Raise 20 Player 2: Raise 25 Player 3: Call
standard output
Correct 210 95 25 110

standard input
3 20 2 50 100 120 AH AC KH KD JD JS JH KS KC AD AS 6 Player 3: Raise 20 Player 1: All in Player 2: Call Player 3: Raise 20 Player 2: Raise 25 Player 3: All in
standard output
Incorrect game log

standard input
3 20 2 50 100 120 AH AC KH KD JD JS JH KS KC AD AS 7 Player 3: Raise 20 Player 1: All in Player 2: Call Player 3: Raise 20 Player 1: All in Player 2: Raise 25 Player 3: Call
standard output
Incorrect game log

standard input
3 20 2 50 100 120 AH AC KH KD JD JS JH KS KC AD AS 6 Player 3: Raise 20 Player 1: All in Player 2: Call Player 3: Raise 20 Player 2: Raise 30 Player 3: Call
standard output
Incorrect game log

standard input
3 20 1 50 80 100 KH QD 2D 3D 7H 2H AS TH JS 4S 5H 8 Player 2: Check Player 3: Check Player 1: Raise 20 Player 2: Call Player 3: Raise 50 Player 1: All in Player 2: All in Player 3: Fold
standard output
Correct 210 50 30 110

Note

Explanation of the last sample: There is a game of three players. The first one has 50 chips, second — 80 and the third one has 100 chips. Let the first player be the *dealer*. Then the second player begins the bets round. The *Blind* size is 20 chips. Then we describe the claims of the players starting form the second player:

1. Player 2: Check
2. Player 3: Check
3. Player 1: Raise 20
4. Player 2: Call
5. Player 3: Raise 50 (Player 3 has bet 70)
6. Player 1: All In (Player 1 doesn't have enough chips so he could either fold or says *All in*)
7. Player 2: All in (Player 2 has already bet 20. Now he calls 50 and wants to raise, but he has only 10 chips and that is not enough for the *Blind* limit. So he goes *All in*)
8. Player 3: Fold

We see there that the bets round ends, because the first player has already moved to the first game (he plays only for 50 chips and no more). The game for the second and the third players (everything more than 50) ends with the third player's fold. The second player wins this game and takes all the chips for this game (all the chips that the third player bet except 50 for the first game). The third player also is excluded from the first game (*Fold* excludes him from all the games). But his 50 chips are still in the bank for the first game. So at this state the bank equals to 150, and the second player has won 20 chips. First player has *Straight* that ends with the Ace and the second player has *Straight* that ends with the five. The highest rank card here is Ace for the first player (the last card for the *Straight*) so the first player wins. Since player 2 was the last aggressor, he opens his cards first, player 3 is not in the game so he doesn't open his cards and the first player wins and opens the cards.

Problem I. Censor

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

HisSql is a top San Francisco startup, known as one of the most exclusive in the area. The startup needs engineers and so it organizes *His[C]up*, a coding competition where the best will be vying for a giant trophy.

Of course, no one has time to prepare for *His[C]up* at *HisSql*. Their laziness is only matched by their arrogance. Instead of coming up with a unique problem, the engineer in charge just reuses a problem he had to solve for work.

The problem reads as follows: *HisSql* keeps track of a database of common words, stored as a "trie". For any query entering *HisSql*, we want to see how many times that query appears as a substring of all the common words.

For example, the strings "ladybug", "ballad", and "ladderlad" might be common words. The query "lad" would then be found 4 times in the list of common words (once in "ladybug", once in "ballad", twice in "ladderlad").

For each query, output the number of times the query appears as a substring in all words of the database.

From Wiki: In computer science, a *trie*, also called digital tree or sometimes radix tree or prefix tree (because we can search by prefixes), is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string.

Input

The first line contains single integer n ($1 \le n \le 10^6$) — number of vertices of trie in the data base excluding the root. Root has index 0. Then n lines follow. The i-th line has a description of i-th vertex in the following format:

- p ($0 \le p \le i - 1$) — parent of the current vertex;
- c (lowercase Latin letter) — character we came to this vertex with from its parent;
- t equals to 1 if this vertex represents the end of some word that ends in this vertex, 0 — otherwise.

The next line contains single integer q — number of queries. Total sum of lengths of all queries is no more than 10^6. Next q lines contains queries (one word per line). All words consist of only lowercase Latin letters and all words are non-empty. All numbers in the input are integers.

Output

For each query output number of times this query is a substring of some word in the data base.

Examples

standard input	standard output
9 0 a 0 1 c 0 2 m 1 1 l 0 4 e 0 5 x 1 0 b 0 7 r 0 8 o 1 3 lex flex ro	1 0 1
5 0 a 0 1 b 0 2 a 1 0 b 0 4 a 1 3 ba a ab	2 3 1

Note

For the first sample the "lex" is the substring of "alex"; and "ro" is a substring of "bro".

Problem J. Rain

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

San Francisco is not a sunny place. If someone tells you that California is a place where everybody is wearing a swim suit every day don't believe. Of course, the weather is not important. The climate is mild, but the winter is rainy in San Francisco. It's funny that ten miles away from San Francisco it is sunny and warm. San Francisco is a magic place. Magic, rainy place.

In this problem we can assume San Francisco could be represented with the rectangular $n \times m$ grid of 1×1 units. Each unit has its own height (San Francisco has hills). On the top and on the bottom of the grid there is an ocean. Rain fills cells starting from the lowest height cell 1 unit per minute. You can assume that the height of the cell is counted as the initial height of the cell plus the water level. The process of filling happens iteratively. Formally, every minute all cells with the minimal height on the current moment are filled with water.

You need to find the first moment when the top of the grid will be connected with the bottom. Each cell is connected up to 4 neighbor cells: for cell (x, y) there are $(x + 1, y)$, $(x - 1, y)$, $(x, y - 1)$, $(x, y + 1)$.

Input

On the first line of the input you are given two integers n, m ($1 \le n, m \le 1000$) — number of rows and columns of the grid. On the next n lines there are m integers on each — height of the corresponded cell — h_ij ($1 \le h_ij \le 10^9$).

Output

Print the first time when oceans will be connected — i.e. there is a path from the top to the bottom of the grid with the cells that are filled with the water.

Examples

standard input	standard output
3 5 102 101 102 10 102 102 100 102 1 102 102 101 102 10 102	10
3 4 1 9 9 9 1 1 5 1 9 9 9 1	5

Problem K. Blinking stars

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

Once walking down the street in San Francisco, professor X realized that there were no clouds in the sky, which was quite unusual for the city. The sky was perfectly clear, and the stars were blinking. He couldn't miss his chance to prove his theory of blinking stars. According to professor X, one star blinks several times in its life. At these moments of time the star produces unbelievable amount of energy. Professor X ran to his apartment to set up all the equipment and start making the most important photo in his life. But when he checked the camera, he realized that he wasted all of the camera's memory on the pictures of his girlfriends. He was not about to delete them. Professor X has only one shot, and this shot should be the best.

To make the shot you need to set up three parameters: width of the frame, height of the frame and exposure (duration when the shutter would be open). Sides of the frame must be parallel to the axes. Professor X wants to have at least k blinks on the picture. All blinks that happen from the start to the end including these moments appear on the picture, including stars located on the border of frame. If one star blinks several times during that period all the times are counted.

Help professor X to find the minimum possible value of frame area multiplied by exposure time to have the possibility of making shot with at least k blinks on his picture.

Input

First line of the input contains two integers n and k ($1 \le n, k \le 50$) — the number of stars and the number of blinks that the professor wants to have on his picture. Then on the following $2n$ lines there is a description of each star (two lines per description). First line for each star contains its coordinates — integers x and y ($-10^5 \le x, y \le 10^5$) and the number of times star blinks — integer m ($1 \le m \le 50$). Next line contains times in milliseconds separated by spaces — integers t ($1 \le t \le 10^5$). Total number of all blinks is no more than 50. No two stars are located at the same point.

Output

Print one number: minimum value of the product of frame width, frame height and exposure duration (in milliseconds). Note, each of the multiples could be equal to zero. If there is no way to get k blinks of the picture print -1.

Examples

standard input	standard output
4 5 0 0 2 1 2 1 1 1 1 1 2 1 1 5 100 1 1	500
3 5 10 10 2 1 2 0 10 2 1 2 10 0 2 2 1	100

Problem L. Cubes (Division 2 only!)

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

In San Francisco there are lots of amazing stores for kids: candies, books and, of course, toys! These stores are always crowded by children who want a toy. But parents are not so fond of buying something just because their kid is crying, they buy toys for Christmas. After one Christmas little Billy found an amazing set of color cubes in the sock.

Billy played with his cubes for a long time. His favorite game was building a wall. He has built a lot of different walls with different lengths and colors. Billy's mother was so proud of her son that she posted photos of all the walls from all directions to her "Instakilogram". Once Billy's mother received message that some of her photos had been gone.

Billy's mother found only two photos on her account that were made from left and from right direction to the wall. To be more specific: if we look at the wall from the front direction we can see the construction with b towers standing next to each other forming the wall. One tower is a set of cubes $1 \times 1 \times h_i$, where h_i is the height of the i-th tower. The height of the tower could be zero. If we look at the wall from one of its side (left or right direction) we can see one tower only (but each cube of the tower could really belong to different towers).

Billy's mother also found the comment under the picture with the quick description: "Billy has built a wall of length no more than b, used cubes of no more than k different colors, the highest tower had height of exactly h". Now she is wondering how many walls could be built by Billy. The number of walls could be huge, so she wants to find the answer modulo 1000000007 ($10^9 + 7$).

Input

The first line of the input contains three integers b, k and h ($1 \le b \le 300, 1 \le k \le 10^5, 1 \le h \le 500$) — the length of the wall, the number of different colors that could be used in the wall (not necessary all of them should be used), and the height of the tower of cubes on the left and right direction picture.

The second line contains the description of the left direction tower: h integers c_i ($1 \le c_i \le k$), where c_i is the color of the i-th cube counting from bottom to top. The third line contains the description of the right direction picture in the same format.

Output

Print the number of walls that could match to these pictures modulo 1000000007 ($10^9 + 7$).

Examples

standard input	standard output
3 3 3 1 2 3 3 2 3	132
2 3 5 1 2 3 2 1 3 2 1 2 1	5
1 1 1 1 1	1

Note

The explanation of the second sample test:

```
11    1    1   1    1
22   22    2   22   2
31   31   31   31   31
22   22   22   22   22
13   13   13   13   13
```

Problem M. Garbage problem (Division 2 only!)

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

The inhabitants of San Francisco have realized that there is a huge garbage problem and someone should immediately start thinking about the solution. But first let's understand what is going on. Scientists concluded that there were several dirty regions in the city and these regions were growing every second. Now the problem that the scientists need to solve is to find out when the garbage areas would separate the city into non-connected components.

In this problem we can assume that the city is a polygon without holes, self intersections and touching. In the beginning each dirty region is a circle with zero radius. All dirty regions are growing with the speed 1, that is at time t radius of each region is t.

You need to figure out the first moment of time when there would be more than one clean region in the city and they would not be connected to each other.

Input

The first line of the input contains integer n ($3 \le n \le 100$) — the number of vertices of the polygon that represents the city. Next n lines contain the points of the polygon in the traverse order.

Next line contains integer k ($k = 1$) — the number of dirty regions. Next k lines represent the centers of dirty regions, one point on each line. Center of each dirty region is either inside the polygon or on the border of the polygon.

All coordinates are integers in range $[-10^5, 10^5]$.

Output

On the first line of the output write "`Never`" if the event never happens, or "`Yes`" if it is possible.

For the "`Yes`" output, second line should contain first moment of time when the city becomes disconnected. Absolute or relative error of the answer must not exceed 10^{-5}.

Examples

standard input	standard output
4 0 0 0 5 9 5 8 0 1 2 4	Yes 2.0
3 0 0 7 2 2 4 1 2 4	Yes 3.2966535347685366

Problem N. Codes (Division 2 Only!)

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

When you communicate over long distances, the bits that you transmit sometimes get accidentally flipped. To deal with this kind of problem, most long-range or wireless communication uses error-correcting codes. The simplest example to understand the concept is as follows: if you want to just send one bit '0' or '1', you could replace it by "000" and "111", respectively. Now, if at most one bit gets flipped in your message, the receiver can still figure out whether your bit was 0 or 1. This kind of code based on duplication is not particularly efficient, and one of the very active areas of research is to design codes that use as few extra bits as possible while allowing as many bit flips as possible.

Here, you will solve a much easier problem. Given a code that someone else has already designed, and a received codeword, you are to figure out how many bits have to have been flipped during transmission. More specifically, you will be given a list of n candidate strings m_i of zeros and ones that would be correct messages, each exactly b bits long. You will also be given the received message r, another bit string of b bits. You are to figure out the minimum number f of bits of r you'd need to flip to get any string m_i.

Input

The first line is the number T ($1 \le T \le 20$) of input data sets, followed by the data sets, each of the following form:

The first line of each data set contains the two integers n and b. This is followed by n lines, each describing one valid codeword as a string of b zeros and ones. After these n lines, there is one more line, containing the string r, again a string of b zeros and ones ($1 \le n \le 1000$, $1 \le b \le 100$).

Output

For each test case output the minimum distance f of the received code from any valid codeword.

Examples

standard input	standard output
2 3 3 000 111 110 010 4 2 00 01 10 11 00	1 0

Problem O. John's Friends (Division 2 Only!)

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

John decides to compare the heights of his friends. John doesn't have appropriate measuring device, so he just compares relative heights between two people (which is easy to do using, for example, a book).

Knowing that none of John's friends are the same height, and John always compares correctly and having results of all of John's comparisons, determine who the tallest person is between two particular John's friends (or if its still unknown).

Input

The first line contains two integers N and M separated by one space. N ($1 \le N \le 10^6$) is the number of John's friends. M ($1 \le M \le 10^7$) is the number of comparisons that have already been done by John. Each of the next M lines contains two distinct integers x and y ($1 \le x, y \le N$) separated by a space, indicating that friend number x was determined to be taller than friend number y. Finally, the last line contains two distinct integers p and q ($1 \le p, q \le N$) separated by one space: your goal is to determine, if possible, whether person p is taller than person q. It is guaranteed that each measurement between any two particular people will be recorded exactly once.

Output

Print "`yes`", if p is taller than q, "`no`" if q is taller than p, and "`unknown`" if John does not collected enough information to answer such a question.

Examples

standard input	standard output
10 3 8 4 3 8 4 2 3 2	yes
10 3 3 8 2 8 3 4 3 2	unknown

Problem O. Query the Black Box (supplementary for F)

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

This problem is supplementary for the problem **F**. It cannot be solved and will return Presentation Error test 1 for any submit.

Here is the Black box description that helps you to understand the format and requirements.

First of all, we tried to make the black box consistent. That means that there is no insane things in the black box.

To make your life easier checker for this problem accepts multiple requests for Black box produced by your program (up to 10). So, for request you must submit a code, printing a request to standard output. To see result of your request, go «Submissions» tab in your ejudge client, then open report for this problem («View» in the «View report» column). In the end of report, after text "`Black box says:`" and empty line will be answer of black box to your requests, in same order as it was sent by your code.

You have limit 100 attempts for this problem, so do not waste them.

To solve the problem **F** you need to repeat the black box logic. The black box you need to implement should accept only one single request and produce a single response. Your black box should take a request, not a code that produces that request.

Input

There is no input for the problem.

Output

On the first line print a single integer n ($1 \le n \le 10$) — number of requests. Next n lines should contain single request per line. Note that request should have length no more than 1000.

Examples

standard input
No input for the problem
standard output

7
bool Alex_Artem_Slava(int, long, short, float, double, void, char);
bool wish_you_good_luck(long long, long double, long int);
bool and_have_fun(unsigned int, unsigned long, unsigned short);
bool and_dont_solve_this_problem(unsigned float, unsigned double);
unsigned float read_other_problems_first(double, long, int, short);
bool if_you_solve_it_we_will_give_you_some_prize(int);
int glhf(unsigned void, unsigned char);