C. Guess the Array

727C

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

*This is an interactive problem. You should use flush operation after each printed line. For example, in C++ you should use fflush(stdout), in Java you should use System.out.flush(), and in Pascal — flush(output).*

In this problem you should guess an array *a* which is unknown for you. The only information you have initially is the length *n* of the array *a*.

The only allowed action is to ask the sum of two elements by their indices. Formally, you can print two indices *i* and *j* (the indices should be **distinct**). Then your program should read the response: the single integer equals to *ai* + *aj*.

It is easy to prove that it is always possible to guess the array using at most *n* requests.

Write a program that will guess the array *a* by making at most *n* requests.

**Interaction**

In each test your program should guess a single array.

The input starts with a line containing integer *n* (3 ≤ *n* ≤ 5000) — the length of the array. Your program should read it at first.

After that your program should print to the standard output the requests about the sum of two elements or inform that the array is guessed.

* In case your program is making a request to ask the sum of two elements, it should print line in the format "? i j" (*i* and *j* are distinct integers between 1 and *n*), where *i* and *j* are indices in the array *a*.
* In case your program informs that the array is guessed, it should print line in the format "! *a*1 *a*2 ... *an*" (it is guaranteed that all *ai*are positive integers not exceeding 105), where *ai* is the *i*-th element of the array *a*.

The response on a request is a single integer equal to *ai* + *aj*, printed on a separate line.

Your program can do at most *n* requests. Note that the final line «! *a*1 *a*2 ... *an*» is not counted as a request.

Do not forget about flush operation after each printed line.

After you program prints the guessed array, it should terminate normally.

**Example**

**input**

5  
   
9  
   
7  
   
9  
   
11  
   
6

**output**

? 1 5  
   
? 2 3  
   
? 4 1  
   
? 5 2  
   
? 3 4  
   
! 4 6 1 5 5

**Note**

The format of a test to make a hack is:

* The first line contains an integer number *n* (3 ≤ *n* ≤ 5000) — the length of the array.
* The second line contains *n* numbers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 105) — the elements of the array to guess.

B. Bill Total Value

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasily exited from a store and now he wants to recheck the total price of all purchases in his bill. The bill is a string in which the names of the purchases and their prices are printed in a row without any spaces. Check has the format "*name*1*price*1*name*2*price*2...*namenpricen*", where *namei* (name of the *i*-th purchase) is a non-empty string of length not more than 10, consisting of lowercase English letters, and *pricei* (the price of the *i*-th purchase) is a non-empty string, consisting of digits and dots (decimal points). It is possible that purchases with equal names have different prices.

The price of each purchase is written in the following format. If the price is an integer number of dollars then cents are not written.

Otherwise, after the number of dollars a dot (decimal point) is written followed by cents **in a two-digit format** (if number of cents is between 1 and 9 inclusively, there is a leading zero).

Also, every three digits (from less significant to the most) in dollars are separated by dot (decimal point). No extra leading zeroes are allowed. The price always starts with a digit and ends with a digit.

For example:

* "234", "1.544", "149.431.10", "0.99" and "123.05" are valid prices,
* ".333", "3.33.11", "12.00", ".33", "0.1234" and "1.2" are not valid.

Write a program that will find the total price of all purchases in the given bill.

**Input**

The only line of the input contains a non-empty string *s* with length not greater than 1000 — the content of the bill.

It is guaranteed that the bill meets the format described above. It is guaranteed that each price in the bill is not less than one cent and not greater than 106 dollars.

**Output**

Print the total price **exactly in the same format** as prices given in the input.

**Examples**

**input**

chipsy48.32televizor12.390

**output**

12.438.32

**input**

a1b2c3.38

**output**

6.38

**input**

aa0.01t0.03

**output**

0.04

A. Transformation: from A to B

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasily has a number *a*, which he wants to turn into a number *b*. For this purpose, he can do two types of operations:

* multiply the current number by 2 (that is, replace the number *x* by 2·*x*);
* append the digit 1 to the right of current number (that is, replace the number *x* by 10·*x* + 1).

You need to help Vasily to transform the number *a* into the number *b* using only the operations described above, or find that it is impossible.

Note that in this task you are not required to minimize the number of operations. It suffices to find any way to transform *a* into *b*.

**Input**

The first line contains two positive integers *a* and *b* (1 ≤ *a* < *b* ≤ 109) — the number which Vasily has and the number he wants to have.

**Output**

If there is no way to get *b* from *a*, print "NO" (without quotes).

Otherwise print three lines. On the first line print "YES" (without quotes). The second line should contain single integer *k* — the length of the transformation sequence. On the third line print the sequence of transformations *x*1, *x*2, ..., *xk*, where:

* *x*1 should be equal to *a*,
* *xk* should be equal to *b*,
* *xi* should be obtained from *xi*- 1 using any of two described operations (1 < *i* ≤ *k*).

If there are multiple answers, print any of them.

**Examples**

**input**

2 162

**output**

YES  
5  
2 4 8 81 162

**input**

4 42

**output**

NO

**input**

100 40021

**output**

YES  
5  
100 200 2001 4002 40021

G. Messages on a Tree

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Alice and Bob are well-known for sending messages to each other. This time you have a rooted tree with Bob standing in the root node and copies of Alice standing in each of the other vertices. The root node has number 0, the rest are numbered 1 through *n*.

At some moments of time some copies of Alice want to send a message to Bob and receive an answer. We will call this copy the *initiator*. The process of sending a message contains several steps:

* The initiator sends the message to the person standing in the parent node and begins waiting for the answer.
* When some copy of Alice receives a message from some of her children nodes, she sends the message to the person standing in the parent node and begins waiting for the answer.
* When Bob receives a message from some of his child nodes, he immediately sends the answer to the child node where the message came from.
* When some copy of Alice (except for initiator) receives an answer she is waiting for, she immediately sends it to the child vertex where the message came from.
* When the initiator receives the answer she is waiting for, she doesn't send it to anybody.
* There is a special case: a copy of Alice can't wait for two answers at the same time, so if some copy of Alice receives a message from her child node while she already waits for some answer, she rejects the message and sends a message saying this back to the child node where the message came from. Then the copy of Alice in the child vertex processes this answer as if it was from Bob.
* The process of sending a message to a parent node or to a child node is instant but a receiver (a parent or a child) gets a message after 1 second.

If some copy of Alice receives several messages from child nodes at the same moment while she isn't waiting for an answer, she processes the message from the **initiator** with the smallest number and rejects all the rest. If some copy of Alice receives messages from children nodes and also receives the answer she is waiting for at the same instant, then Alice first processes the answer, then immediately continue as normal with the incoming messages.

You are given the moments of time when some copy of Alice becomes the initiator and sends a message to Bob. For each message, find the moment of time when the answer (either from Bob or some copy of Alice) will be received by the initiator.

You can assume that if Alice wants to send a message (i.e. become the initiator) while waiting for some answer, she immediately rejects the message and receives an answer from herself in no time.

**Input**

The first line of input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 200 000) — the number of nodes with Alices and the number of messages.

Second line contains *n* integers *p*1, *p*2, ..., *pn* (0 ≤ *pi* < *i*). The integer *pi* is the number of the parent node of node *i*.

The next *m* lines describe the messages. The *i*-th of them contains two integers *xi* and *ti* (1 ≤ *xi* ≤ *n*, 1 ≤ *ti* ≤ 109) — the number of the vertex of the initiator of the *i*-th message and the time of the initiation (in seconds). The messages are given in order of increasing initiation time (i.e. *ti*+ 1 ≥ *ti* holds for 1 ≤ *i* < *m*). The pairs (*xi*, *ti*) are distinct.

**Output**

Print *m* integers — the *i*-th of them is the moment of time when the answer for the *i*-th message will be received by the initiator.

**Examples**

**input**

6 3  
0 1 2 3 2 5  
4 6  
6 9  
5 11

**output**

14 13 11

**input**

3 2  
0 1 1  
2 1  
3 1

**output**

5 3

**input**

8 3  
0 1 1 2 3 3 4 5  
6 1  
8 2  
4 5

**output**

7 6 11

**Note**

In the first example the first message is initiated at the moment 6, reaches Bob at the moment 10, and the answer reaches the initiator at the moment 14. The second message reaches vertex 2 at the moment 11. At this moment the copy of Alice in this vertex is still waiting for the answer for the first message, so she rejects the second message. The answer reaches the initiator at the moment 13. The third message is not sent at all, because at the moment 11 Alice in vertex 5 is waiting for the answer for the second message.

In the second example the first message reaches Bob, the second is rejected by Alice in vertex 1. This is because the message with smaller initiator number has the priority.

In the third example the first and the third messages reach Bob, while the second message is rejected by Alice in vertex 3.

F. Family Photos

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Alice and Bonnie are sisters, but they don't like each other very much. So when some old family photos were found in the attic, they started to argue about who should receive which photos. In the end, they decided that they would take turns picking photos. Alice goes first.

There are *n* stacks of photos. Each stack contains **exactly two** photos. In each turn, a player may take only a photo from the top of one of the stacks.

Each photo is described by two non-negative integers *a* and *b*, indicating that it is worth *a* units of happiness to Alice and *b* units of happiness to Bonnie. Values of *a* and *b* might differ for different photos.

It's allowed to pass instead of taking a photo. The game ends when all photos are taken or both players pass consecutively.

The players don't act to maximize their own happiness. Instead, each player acts to maximize the amount by which her happiness exceeds her sister's. Assuming both players play optimal, find the difference between Alice's and Bonnie's happiness. That is, if there's a perfectly-played game such that Alice has *x* happiness and Bonnie has *y* happiness at the end, you should print *x* - *y*.

**Input**

The first line of input contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the number of two-photo stacks. Then follow *n* lines, each describing one of the stacks. A stack is described by four space-separated non-negative integers *a*1, *b*1, *a*2 and *b*2, each not exceeding 109. *a*1 and *b*1 describe the top photo in the stack, while *a*2 and *b*2 describe the bottom photo in the stack.

**Output**

Output a single integer: the difference between Alice's and Bonnie's happiness if both play optimally.

**Examples**

**input**

2  
12 3 4 7  
1 15 9 1

**output**

1

**input**

2  
5 4 8 8  
4 12 14 0

**output**

4

**input**

1  
0 10 0 10

**output**

-10

E. Too Much Money

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Alfred wants to buy a toy moose that costs *c* dollars. The store doesn’t give change, so he must give the store exactly *c* dollars, no more and no less. He has *n* coins. To make *c* dollars from his coins, he follows the following algorithm: let *S* be the set of coins being used. *S*is initially empty. Alfred repeatedly adds to *S* the highest-valued coin he has such that the total value of the coins in *S* after adding the coin doesn’t exceed *c*. If there is no such coin, and the value of the coins in *S* is still less than *c*, he gives up and goes home. Note that Alfred never removes a coin from *S* after adding it.

As a programmer, you might be aware that Alfred’s algorithm can fail even when there is a set of coins with value exactly *c*. For example, if Alfred has one coin worth $3, one coin worth $4, and two coins worth $5, and the moose costs $12, then Alfred will add both of the $5 coins to *S* and then give up, since adding any other coin would cause the value of the coins in *S* to exceed $12. Of course, Alfred could instead combine one $3 coin, one $4 coin, and one $5 coin to reach the total.

Bob tried to convince Alfred that his algorithm was flawed, but Alfred didn’t believe him. Now Bob wants to give Alfred some coins (in addition to those that Alfred already has) such that Alfred’s algorithm fails. Bob can give Alfred any number of coins of any denomination (subject to the constraint that each coin must be worth a positive integer number of dollars). There can be multiple coins of a single denomination. He would like to minimize the total value of the coins he gives Alfred. Please find this minimum value. If there is no solution, print "Greed is good". You can assume that the answer, if it exists, is positive. In other words, Alfred's algorithm will work if Bob doesn't give him any coins.

**Input**

The first line contains *c* (1 ≤ *c* ≤ 200 000) — the price Alfred wants to pay. The second line contains *n* (1 ≤ *n* ≤ 200 000) — the number of coins Alfred initially has. Then *n* lines follow, each containing a single integer *x* (1 ≤ *x* ≤ *c*) representing the value of one of Alfred's coins.

**Output**

If there is a solution, print the minimum possible total value of the coins in a solution. Otherwise, print "Greed is good" (without quotes).

**Examples**

**input**

12  
3  
5  
3  
4

**output**

5

**input**

50  
8  
1  
2  
4  
8  
16  
37  
37  
37

**output**

Greed is good

**Note**

In the first sample, Bob should give Alfred a single coin worth $5. This creates the situation described in the problem statement.

In the second sample, there is no set of coins that will cause Alfred's algorithm to fail.

D. Contest Balloons

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

One tradition of ACM-ICPC contests is that a team gets a balloon for every solved problem. We assume that the submission time doesn't matter and teams are sorted only by the number of balloons they have. It means that one's place is equal to the number of teams with more balloons, increased by 1. For example, if there are seven teams with more balloons, you get the eight place. Ties are allowed.

You should know that it's important to eat before a contest. If the number of balloons of a team is greater than the weight of this team, the team starts to float in the air together with their workstation. They eventually touch the ceiling, what is strictly forbidden by the rules. The team is then disqualified and isn't considered in the standings.

A contest has just finished. There are *n* teams, numbered 1 through *n*. The *i*-th team has *ti* balloons and weight *wi*. It's guaranteed that *ti* doesn't exceed *wi* so nobody floats initially.

Limak is a member of the first team. He doesn't like cheating and he would never steal balloons from other teams. Instead, he can give his balloons away to other teams, possibly making them float. Limak can give away zero or more balloons of his team. Obviously, he can't give away more balloons than his team initially has.

What is the best place Limak can get?

**Input**

The first line of the standard input contains one integer *n* (2 ≤ *n* ≤ 300 000) — the number of teams.

The *i*-th of *n* following lines contains two integers *ti* and *wi* (0 ≤ *ti* ≤ *wi* ≤ 1018) — respectively the number of balloons and the weight of the *i*-th team. Limak is a member of the first team.

**Output**

Print one integer denoting the best place Limak can get.

**Examples**

**input**

8  
20 1000  
32 37  
40 1000  
45 50  
16 16  
16 16  
14 1000  
2 1000

**output**

3

**input**

7  
4 4  
4 4  
4 4  
4 4  
4 4  
4 4  
5 5

**output**

2

**input**

7  
14000000003 1000000000000000000  
81000000000 88000000000  
5000000000 7000000000  
15000000000 39000000000  
46000000000 51000000000  
0 1000000000  
0 0

**output**

2

**Note**

In the first sample, Limak has 20 balloons initially. There are three teams with more balloons (32, 40 and 45 balloons), so Limak has the fourth place initially. One optimal strategy is:

1. Limak gives 6 balloons away to a team with 32 balloons and weight 37, which is just enough to make them fly. Unfortunately, Limak has only 14 balloons now and he would get the fifth place.
2. Limak gives 6 balloons away to a team with 45 balloons. Now they have 51 balloons and weight 50 so they fly and get disqualified.
3. Limak gives 1 balloon to each of two teams with 16 balloons initially.
4. Limak has 20 - 6 - 6 - 1 - 1 = 6 balloons.
5. There are three other teams left and their numbers of balloons are 40, 14 and 2.
6. Limak gets the third place because there are two teams with more balloons.

In the second sample, Limak has the second place and he can't improve it.

In the third sample, Limak has just enough balloons to get rid of teams 2, 3 and 5 (the teams with 81 000 000 000, 5 000 000 000 and 46 000 000 000 balloons respectively). With zero balloons left, he will get the second place (ex-aequo with team 6 and team 7).

C. Hidden Word

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Let’s define a grid to be a set of tiles with 2 rows and 13 columns. Each tile has an English letter written in it. The letters don't have to be unique: there might be two or more tiles with the same letter written on them. Here is an example of a grid:

ABCDEFGHIJKLM  
NOPQRSTUVWXYZ

We say that two tiles are adjacent if they share a side or a corner. In the example grid above, the tile with the letter 'A' is adjacent only to the tiles with letters 'B', 'N', and 'O'. A tile is not adjacent to itself.

A sequence of tiles is called a path if each tile in the sequence is adjacent to the tile which follows it (except for the last tile in the sequence, which of course has no successor). In this example, "ABC" is a path, and so is "KXWIHIJK". "MAB" is not a path because 'M' is not adjacent to 'A'. A single tile can be used more than once by a path (though the tile cannot occupy two consecutive places in the path because no tile is adjacent to itself).

You’re given a string *s* which consists of 27 upper-case English letters. Each English letter **occurs at least once** in *s*. Find a grid that contains a path whose tiles, viewed in the order that the path visits them, form the string *s*. If there’s no solution, print "Impossible" (without the quotes).

**Input**

The only line of the input contains the string *s*, consisting of 27 upper-case English letters. Each English letter occurs at least once in *s*.

**Output**

Output two lines, each consisting of 13 upper-case English characters, representing the rows of the grid. If there are multiple solutions, print any of them. If there is no solution print "Impossible".

**Examples**

**input**

ABCDEFGHIJKLMNOPQRSGTUVWXYZ

**output**

YXWVUTGHIJKLM  
ZABCDEFSRQPON

**input**

BUVTYZFQSNRIWOXXGJLKACPEMDH

**output**

Impossible

B. Food on the Plane

time limit per test

2 seconds

memory limit per test

256 megabytes

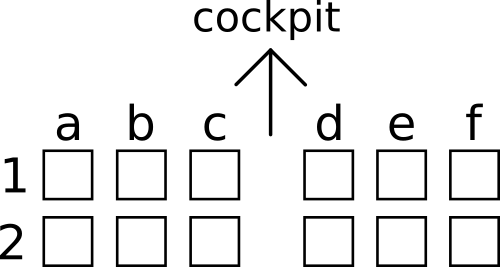
input

standard input

output

standard output

A new airplane SuperPuperJet has an infinite number of rows, numbered with positive integers starting with 1 from cockpit to tail. There are six seats in each row, denoted with letters from 'a' to 'f'. Seats 'a', 'b' and 'c' are located to the left of an aisle (if one looks in the direction of the cockpit), while seats 'd', 'e' and 'f' are located to the right. Seats 'a' and 'f' are located near the windows, while seats 'c' and 'd' are located near the aisle.



It's lunch time and two flight attendants have just started to serve food. They move from the first rows to the tail, always maintaining a distance of two rows from each other because of the food trolley. Thus, at the beginning the first attendant serves row 1 while the second attendant serves row 3. When both rows are done they move one row forward: the first attendant serves row 2 while the second attendant serves row 4. Then they move three rows forward and the first attendant serves row 5 while the second attendant serves row 7. Then they move one row forward again and so on.

Flight attendants work with the same speed: it takes exactly 1 second to serve one passenger and 1 second to move one row forward. Each attendant first serves the passengers on the seats to the right of the aisle and then serves passengers on the seats to the left of the aisle (if one looks in the direction of the cockpit). Moreover, they always serve passengers in order from the window to the aisle. Thus, the first passenger to receive food in each row is located in seat 'f', and the last one — in seat 'c'. Assume that all seats are occupied.

Vasya has seat *s* in row *n* and wants to know how many seconds will pass before he gets his lunch.

**Input**

The only line of input contains a description of Vasya's seat in the format *ns*, where *n* (1 ≤ *n* ≤ 1018) is the index of the row and *s* is the seat in this row, denoted as letter from 'a' to 'f'. The index of the row and the seat **are not separated** by a space.

**Output**

Print one integer — the number of seconds Vasya has to wait until he gets his lunch.

**Examples**

**input**

1f

**output**

1

**input**

2d

**output**

10

**input**

4a

**output**

11

**input**

5e

**output**

18

**Note**

In the first sample, the first flight attendant serves Vasya first, so Vasya gets his lunch after 1 second.

In the second sample, the flight attendants will spend 6 seconds to serve everyone in the rows 1 and 3, then they will move one row forward in 1 second. As they first serve seats located to the right of the aisle in order from window to aisle, Vasya has to wait 3 more seconds. The total is 6 + 1 + 3 = 10.

A. Jumping Ball

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

In a new version of the famous Pinball game, one of the most important parts of the game field is a sequence of *n* bumpers. The bumpers are numbered with integers from 1 to *n* from left to right. There are two types of bumpers. They are denoted by the characters '<' and '>'. When the ball hits the bumper at position *i* it goes one position to the right (to the position *i* + 1) if the type of this bumper is '>', or one position to the left (to *i* - 1) if the type of the bumper at position *i* is '<'. If there is no such position, in other words if *i* - 1 < 1 or *i* + 1 > *n*, the ball falls from the game field.

Depending on the ball's starting position, the ball may eventually fall from the game field or it may stay there forever. You are given a string representing the bumpers' types. Calculate the number of positions such that the ball will eventually fall from the game field if it starts at that position.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 200 000) — the length of the sequence of bumpers. The second line contains the string, which consists of the characters '<' and '>'. The character at the *i*-th position of this string corresponds to the type of the *i*-th bumper.

**Output**

Print one integer — the number of positions in the sequence such that the ball will eventually fall from the game field if it starts at that position.

**Examples**

**input**

4  
<<><

**output**

2

**input**

5  
>>>>>

**output**

5

**input**

4  
>><<

**output**

0

**Note**

In the first sample, the ball will fall from the field if starts at position 1 or position 2.

In the second sample, any starting position will result in the ball falling from the field.

G. Xor-matic Number of the Graph

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given an undirected graph, constisting of *n* vertices and *m* edges. Each edge of the graph has some non-negative integer written on it.

Let's call a triple (*u*, *v*, *s*) **interesting**, if 1 ≤ *u* < *v* ≤ *n* and there is a path (**possibly non-simple**, i.e. it can visit the same vertices and edges multiple times) between vertices *u* and *v* such that xor of all numbers written on the edges of this path is equal to *s*. **When we compute the value s for some path, each edge is counted in xor as many times, as it appear on this path.** It's not hard to prove that there are finite number of such triples.

Calculate the sum over modulo 109 + 7 of the values of *s* over all **interesting** triples.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n* ≤ 100 000, 0 ≤ *m* ≤ 200 000) — numbers of vertices and edges in the given graph.

The follow *m* lines contain three integers *ui*, *vi* and *ti* (1 ≤ *ui*, *vi* ≤ *n*, 0 ≤ *ti* ≤ 1018, *ui* ≠ *vi*) — vertices connected by the edge and integer written on it. It is guaranteed that graph doesn't contain self-loops and multiple edges.

**Output**

Print the single integer, equal to the described sum over modulo 109 + 7.

**Examples**

**input**

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

**output**

12

**input**

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

**output**

90

**input**

8 6  
1 2 2  
2 3 1  
2 4 4  
4 5 5  
4 6 3  
7 8 5

**output**

62

**Note**

In the first example the are 6 interesting triples:

1. (1, 2, 1)
2. (1, 3, 2)
3. (1, 4, 3)
4. (2, 3, 3)
5. (2, 4, 2)
6. (3, 4, 1)

The sum is equal to 1 + 2 + 3 + 3 + 2 + 1 = 12.

In the second example the are 12 interesting triples:

1. (1, 2, 1)
2. (2, 3, 2)
3. (1, 3, 3)
4. (3, 4, 4)
5. (2, 4, 6)
6. (1, 4, 7)
7. (1, 4, 8)
8. (2, 4, 9)
9. (3, 4, 11)
10. (1, 3, 12)
11. (2, 3, 13)
12. (1, 2, 14)

The sum is equal to 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9 + 11 + 12 + 13 + 14 = 90.

F. Uniformly Branched Trees

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

A tree is a connected graph without cycles.

Two trees, consisting of *n* vertices each, are called *isomorphic* if there exists a permutation *p*: {1, ..., *n*} → {1, ..., *n*} such that the edge (*u*, *v*) is present in the first tree if and only if the edge (*pu*, *pv*) is present in the second tree.

Vertex of the tree is called internal if its degree is greater than or equal to two.

Count the number of different non-isomorphic trees, consisting of *n* vertices, such that the degree of each internal vertex is **exactly** *d*. Print the answer over the given prime modulo *mod*.

**Input**

The single line of the input contains three integers *n*, *d* and *mod* (1 ≤ *n* ≤ 1000, 2 ≤ *d* ≤ 10, 108 ≤ *mod* ≤ 109)  — the number of vertices in the tree, the degree of internal vertices and the prime modulo.

**Output**

Print the number of trees over the modulo *mod*.

**Examples**

**input**

5 2 433416647

**output**

1

**input**

10 3 409693891

**output**

2

**input**

65 4 177545087

**output**

910726

E. Goods transportation

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

There are *n* cities located along the one-way road. Cities are numbered from 1 to *n* in the direction of the road.

The *i*-th city had produced *pi* units of goods. No more than *si* units of goods can be sold in the *i*-th city.

For each pair of cities *i* and *j* such that **1 ≤ *i* < *j* ≤ *n*** you can **no more than once** transport **no more than** *c* units of goods from the city *i* to the city *j*. Note that goods can only be transported from a city with a lesser index to the city with a larger index. **You can transport goods between cities in any order.**

Determine the maximum number of produced goods that can be sold in total in all the cities after a sequence of transportations.

**Input**

The first line of the input contains two integers *n* and *c* (1 ≤ *n* ≤ 10 000, 0 ≤ *c* ≤ 109) — the number of cities and the maximum amount of goods for a single transportation.

The second line contains *n* integers *pi* (0 ≤ *pi* ≤ 109) — the number of units of goods that were produced in each city.

The third line of input contains *n* integers *si* (0 ≤ *si* ≤ 109) — the number of units of goods that can be sold in each city.

**Output**

Print the maximum total number of produced goods that can be sold in all cities after a sequence of transportations.

**Examples**

**input**

3 0  
1 2 3  
3 2 1

**output**

4

**input**

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

**output**

12

**input**

4 3  
13 10 7 4  
4 7 10 13

**output**

34

D. Dense Subsequence

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a string *s*, consisting of lowercase English letters, and the integer *m*.

One should choose some symbols from the given string so that any contiguous subsegment of length *m* has at least one selected symbol. Note that here we choose positions of symbols, not the symbols themselves.

Then one uses the chosen symbols to form **a new string**. All symbols from the chosen position should be used, but we are allowed to rearrange them in any order.

Formally, we choose a subsequence of indices 1 ≤ *i*1 < *i*2 < ... < *it* ≤ |*s*|. The selected sequence must meet the following condition: for every *j* such that 1 ≤ *j* ≤ |*s*| - *m* + 1, there must be at least one selected index that belongs to the segment [*j*,  *j* + *m* - 1], i.e. there should exist a *k* from 1 to *t*, such that *j* ≤ *ik* ≤ *j* + *m* - 1.

Then we take any permutation *p* of the selected indices and form a new string *sip*1*sip*2... *sipt*.

Find the lexicographically smallest string, that can be obtained using this procedure.

**Input**

The first line of the input contains a single integer *m* (1 ≤ *m* ≤ 100 000).

The second line contains the string *s* consisting of lowercase English letters. It is guaranteed that this string is non-empty and its length doesn't exceed 100 000. It is also guaranteed that the number *m* doesn't exceed the length of the string *s*.

**Output**

Print the single line containing the lexicographically smallest string, that can be obtained using the procedure described above.

**Examples**

**input**

3  
cbabc

**output**

a

**input**

2  
abcab

**output**

aab

**input**

3  
bcabcbaccba

**output**

aaabb

**Note**

In the first sample, one can choose the subsequence {3} and form a string "a".

In the second sample, one can choose the subsequence {1, 2, 4} (symbols on this positions are 'a', 'b' and 'a') and rearrange the chosen symbols to form a string "aab".

C. Ray Tracing

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

There are *k* sensors located in the rectangular room of size *n* × *m* meters. The *i*-th sensor is located at point (*xi*, *yi*). All sensors are located at distinct points strictly inside the rectangle.

Opposite corners of the room are located at points (0, 0) and (*n*, *m*). Walls of the room are parallel to coordinate axes.

At the moment 0, from the point (0, 0) the laser ray is released in the direction of point (1, 1). The ray travels with a speed of http://codeforces.com/predownloaded/2a/b5/2ab51c90d460da0abc5c722c35772dc30203dd91.png meters per second. Thus, the ray will reach the point (1, 1) in exactly one second after the start.

When the ray meets the wall it's reflected by the rule that the angle of incidence is equal to the angle of reflection. If the ray reaches any of the four corners, it immediately stops.

For each sensor you have to determine the first moment of time when the ray will pass through the point where this sensor is located. If the ray will never pass through this point, print  - 1 for such sensors.

**Input**

The first line of the input contains three integers *n*, *m* and *k* (2 ≤ *n*, *m* ≤ 100 000, 1 ≤ *k* ≤ 100 000) — lengths of the room's walls and the number of sensors.

Each of the following *k* lines contains two integers *xi* and *yi* (1 ≤ *xi* ≤ *n* - 1, 1 ≤ *yi* ≤ *m* - 1) — coordinates of the sensors. It's guaranteed that no two sensors are located at the same point.

**Output**

Print *k* integers. The *i*-th of them should be equal to the number of seconds when the ray first passes through the point where the *i*-th sensor is located, or  - 1 if this will never happen.

**Examples**

**input**

3 3 4  
1 1  
1 2  
2 1  
2 2

**output**

1  
-1  
-1  
2

**input**

3 4 6  
1 1  
2 1  
1 2  
2 2  
1 3  
2 3

**output**

1  
-1  
-1  
2  
5  
-1

**input**

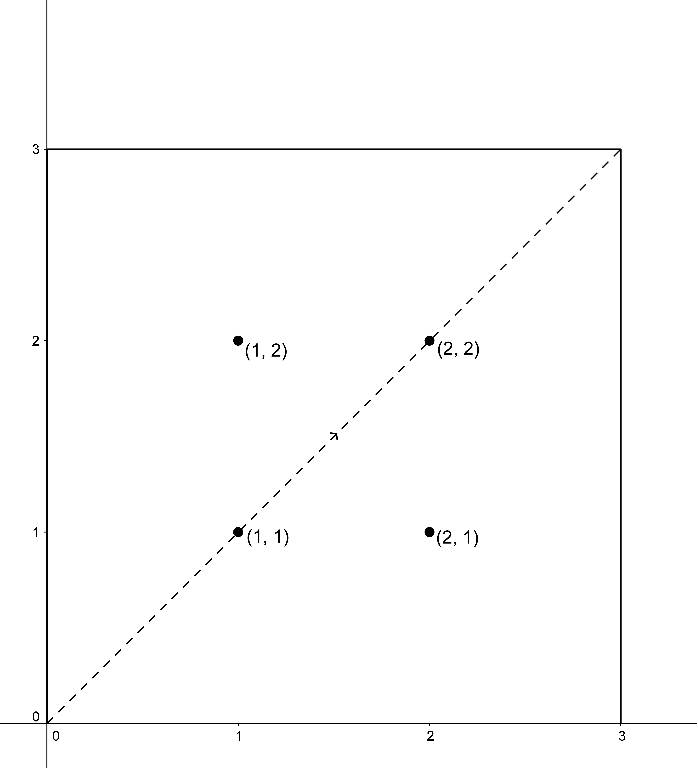
7 4 5  
1 3  
2 2  
5 1  
5 3  
4 3

**output**

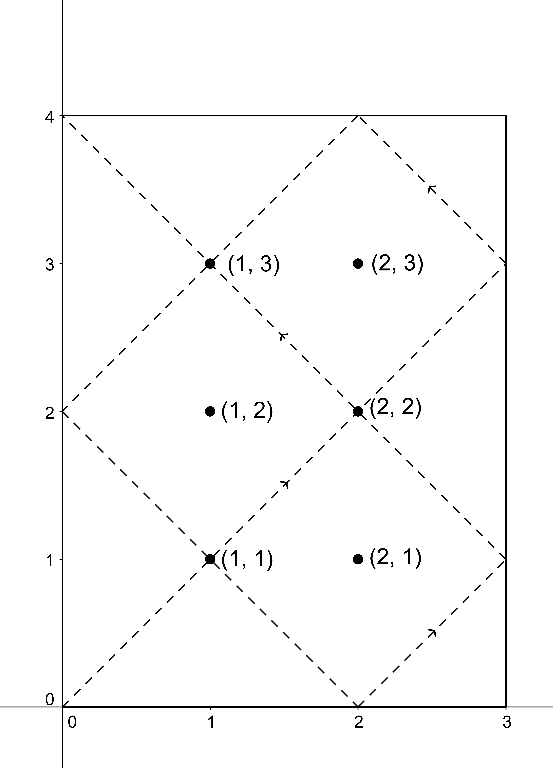
13  
2  
9  
5  
-1

**Note**

In the first sample, the ray will consequently pass through the points (0, 0), (1, 1), (2, 2), (3, 3). Thus, it will stop at the point (3, 3) after 3 seconds.



In the second sample, the ray will consequently pass through the following points: (0, 0), (1, 1), (2, 2), (3, 3), (2, 4), (1, 3), (0, 2), (1, 1), (2, 0), (3, 1), (2, 2), (1, 3), (0, 4). The ray will stop at the point (0, 4) after 12 seconds. It will reflect at the points (3, 3), (2, 4), (0, 2), (2, 0) and (3, 1).



B. Batch Sort

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a table consisting of *n* rows and *m* columns.

Numbers in each row form a permutation of integers from 1 to *m*.

You are allowed to pick two elements in one row and swap them, but **no more than once** for each row. Also, **no more than once** you are allowed to pick two columns and swap them. Thus, you are allowed to perform from 0 to *n* + 1 actions in total. **Operations can be performed in any order**.

You have to check whether it's possible to obtain the identity permutation 1, 2, ..., *m* in each row. In other words, check if one can perform some of the operation following the given rules and make each row sorted in increasing order.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 20) — the number of rows and the number of columns in the given table.

Each of next *n* lines contains *m* integers — elements of the table. It's guaranteed that numbers in each line form a permutation of integers from 1 to *m*.

**Output**

If there is a way to obtain the identity permutation in each row by following the given rules, print "YES" (without quotes) in the only line of the output. Otherwise, print "NO" (without quotes).

**Examples**

**input**

2 4  
1 3 2 4  
1 3 4 2

**output**

YES

**input**

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

**output**

NO

**input**

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

**output**

YES

**Note**

In the first sample, one can act in the following way:

1. Swap second and third columns. Now the table is

1 2 3 4

1 4 3 2

1. In the second row, swap the second and the fourth elements. Now the table is

1 2 3 4

1 2 3 4

A. Checking the Calendar

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given names of two days of the week.

Please, determine whether it is possible that during some **non-leap year** the first day of some month was equal to the first day of the week you are given, while the first day of **the next month** was equal to the second day of the week you are given. **Both months should belong to one year**.

In this problem, we consider the Gregorian calendar to be used. The number of months in this calendar is equal to 12. The number of days in months during any non-leap year is: 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31.

Names of the days of the week are given with lowercase English letters: "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday".

**Input**

The input consists of two lines, each of them containing the name of exactly one day of the week. It's guaranteed that each string in the input is from the set "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday".

**Output**

Print "YES" (without quotes) if such situation is possible during some non-leap year. Otherwise, print "NO" (without quotes).

**Examples**

**input**

monday  
tuesday

**output**

NO

**input**

sunday  
sunday

**output**

YES

**input**

saturday  
tuesday

**output**

YES

**Note**

In the second sample, one can consider February 1 and March 1 of year 2015. Both these days were Sundays.

In the third sample, one can consider July 1 and August 1 of year 2017. First of these two days is Saturday, while the second one is Tuesday.

F. st-Spanning Tree

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given an undirected connected graph consisting of *n* vertices and *m* edges. There are no loops and no multiple edges in the graph.

You are also given two distinct vertices *s* and *t*, and two values *ds* and *dt*. Your task is to build any spanning tree of the given graph (note that the graph is not weighted), such that the degree of the vertex *s* doesn't exceed *ds*, and the degree of the vertex *t* doesn't exceed *dt*, or determine, that there is no such spanning tree.

The *spanning tree* of the graph *G* is a subgraph which is a tree and contains all vertices of the graph *G*. In other words, it is a connected graph which contains *n* - 1 edges and can be obtained by removing some of the edges from *G*.

The degree of a vertex is the number of edges incident to this vertex.

**Input**

The first line of the input contains two integers *n* and *m* (2 ≤ *n* ≤ 200 000, 1 ≤ *m* ≤ *min*(400 000, *n*·(*n* - 1) / 2)) — the number of vertices and the number of edges in the graph.

The next *m* lines contain the descriptions of the graph's edges. Each of the lines contains two integers *u* and *v* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*) — the ends of the corresponding edge. It is guaranteed that the graph contains no loops and no multiple edges and that it is connected.

The last line contains four integers *s*, *t*, *ds*, *dt* (1 ≤ *s*, *t* ≤ *n*, *s* ≠ *t*, 1 ≤ *ds*, *dt* ≤ *n* - 1).

**Output**

If the answer doesn't exist print "No" (without quotes) in the only line of the output.

Otherwise, in the first line print "Yes" (without quotes). In the each of the next (*n* - 1) lines print two integers — the description of the edges of the spanning tree. Each of the edges of the spanning tree must be printed exactly once.

You can output edges in any order. You can output the ends of each edge in any order.

If there are several solutions, print any of them.

**Examples**

**input**

3 3  
1 2  
2 3  
3 1  
1 2 1 1

**output**

Yes  
3 2  
1 3

**input**

7 8  
7 4  
1 3  
5 4  
5 7  
3 2  
2 4  
6 1  
1 2  
6 4 1 4

**output**

Yes  
1 3  
5 7  
3 2  
7 4  
2 4  
6 1

E. One-Way Reform

723E

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

There are *n* cities and *m* two-way roads in Berland, each road connects two cities. It is known that there is no more than one road connecting each pair of cities, and there is no road which connects the city with itself. It is possible that there is no way to get from one city to some other city using only these roads.

The road minister decided to make a reform in Berland and to orient all roads in the country, i.e. to make each road one-way. The minister wants to **maximize** the number of cities, for which the number of roads that begins in the city **equals** to the number of roads that ends in it.

**Input**

The first line contains a positive integer *t* (1 ≤ *t* ≤ 200) — the number of testsets in the input.

Each of the testsets is given in the following way. The first line contains two integers *n* and *m* (1 ≤ *n* ≤ 200, 0 ≤ *m* ≤ *n*·(*n* - 1) / 2) — the number of cities and the number of roads in Berland.

The next *m* lines contain the description of roads in Berland. Each line contains two integers *u* and *v* (1 ≤ *u*, *v* ≤ *n*) — the cities the corresponding road connects. It's guaranteed that there are no self-loops and multiple roads. It is possible that there is no way along roads between a pair of cities.

It is guaranteed that the total number of cities in all testset of input data doesn't exceed 200.

Pay attention that for **hacks**, you can only use tests consisting of **one testset**, so *t* should be equal to one.

**Output**

For each testset print the maximum number of such cities that the number of roads that begins in the city, is equal to the number of roads that ends in it.

In the next *m* lines print oriented roads. First print the number of the city where the road begins and then the number of the city where the road ends. If there are several answers, print any of them. It is allowed to print roads in each test in arbitrary order. Each road should be printed exactly once.

**Example**

**input**

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

**output**

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

D. Lakes in Berland

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

The map of Berland is a rectangle of the size *n* × *m*, which consists of cells of size 1 × 1. Each cell is either land or water. The map is surrounded by the ocean.

*Lakes* are the maximal regions of water cells, connected by sides, which are not connected with the ocean. Formally, lake is a set of water cells, such that it's possible to get from any cell of the set to any other without leaving the set and moving only to cells adjacent by the side, none of them is located on the border of the rectangle, and it's impossible to add one more water cell to the set such that it will be connected with any other cell.

You task is to fill up with the earth the minimum number of water cells so that there will be **exactly** *k* lakes in Berland. Note that the initial number of lakes on the map is **not less** than *k*.

**Input**

The first line of the input contains three integers *n*, *m* and *k* (1 ≤ *n*, *m* ≤ 50, 0 ≤ *k* ≤ 50) — the sizes of the map and the number of lakes which should be left on the map.

The next *n* lines contain *m* characters each — the description of the map. Each of the characters is either '.' (it means that the corresponding cell is water) or '\*' (it means that the corresponding cell is land).

It is guaranteed that the map contain at least *k* lakes.

**Output**

In the first line print the minimum number of cells which should be transformed from water to land.

In the next *n* lines print *m* symbols — the map after the changes. The format must strictly follow the format of the map in the input data (there is no need to print the size of the map). If there are several answers, print any of them.

It is guaranteed that the answer exists on the given data.

**Examples**

**input**

5 4 1  
\*\*\*\*  
\*..\*  
\*\*\*\*  
\*\*.\*  
..\*\*

**output**

1  
\*\*\*\*  
\*..\*  
\*\*\*\*  
\*\*\*\*  
..\*\*

**input**

3 3 0  
\*\*\*  
\*.\*  
\*\*\*

**output**

1  
\*\*\*  
\*\*\*  
\*\*\*

**Note**

In the first example there are only two lakes — the first consists of the cells (2, 2) and (2, 3), the second consists of the cell (4, 3). It is profitable to cover the second lake because it is smaller. Pay attention that the area of water in the lower left corner is not a lake because this area share a border with the ocean.

C. Polycarp at the Radio

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Polycarp is a music editor at the radio station. He received a playlist for tomorrow, that can be represented as a sequence *a*1, *a*2, ..., *an*, where *ai* is a band, which performs the *i*-th song. Polycarp likes bands with the numbers from 1 to *m*, but he doesn't really like others.

We define as *bj* the number of songs the group *j* is going to perform tomorrow. Polycarp wants to change the playlist in such a way that the minimum among the numbers *b*1, *b*2, ..., *bm* will be as large as possible.

Find this maximum possible value of the minimum among the *bj* (1 ≤ *j* ≤ *m*), and the minimum number of changes in the playlist Polycarp needs to make to achieve it. One change in the playlist is a replacement of the performer of the *i*-th song with any other group.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *m* ≤ *n* ≤ 2000).

The second line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109), where *ai* is the performer of the *i*-th song.

**Output**

In the first line print two integers: the maximum possible value of the minimum among the *bj* (1 ≤ *j* ≤ *m*), where *bj* is the number of songs in the changed playlist performed by the *j*-th band, and the minimum number of changes in the playlist Polycarp needs to make.

In the second line print the changed playlist.

If there are multiple answers, print any of them.

**Examples**

**input**

4 2  
1 2 3 2

**output**

2 1  
1 2 1 2

**input**

7 3  
1 3 2 2 2 2 1

**output**

2 1  
1 3 3 2 2 2 1

**input**

4 4  
1000000000 100 7 1000000000

**output**

1 4  
1 2 3 4

**Note**

In the first sample, after Polycarp's changes the first band performs two songs (*b*1 = 2), and the second band also performs two songs (*b*2 = 2). Thus, the minimum of these values equals to 2. It is impossible to achieve a higher minimum value by any changes in the playlist.

In the second sample, after Polycarp's changes the first band performs two songs (*b*1 = 2), the second band performs three songs (*b*2 = 3), and the third band also performs two songs (*b*3 = 2). Thus, the best minimum value is 2.

B. Text Document Analysis

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Modern text editors usually show some information regarding the document being edited. For example, the number of words, the number of pages, or the number of characters.

In this problem you should implement the similar functionality.

You are given a string which only consists of:

* uppercase and lowercase English letters,
* underscore symbols (they are used as separators),
* parentheses (both opening and closing).

It is guaranteed that each opening parenthesis has a succeeding closing parenthesis. Similarly, each closing parentheses has a preceding opening parentheses matching it. For each pair of matching parentheses there are no other parenthesis between them. In other words, each parenthesis in the string belongs to a matching "opening-closing" pair, and such pairs can't be nested.

For example, the following string is valid: "\_Hello\_Vasya(and\_Petya)\_\_bye\_(and\_OK)".

*Word* is a maximal sequence of consecutive letters, i.e. such sequence that the first character to the left and the first character to the right of it is an underscore, a parenthesis, or it just does not exist. For example, the string above consists of seven words: "Hello", "Vasya", "and", "Petya", "bye", "and" and "OK". Write a program that finds:

* the length of the longest word outside the parentheses (print 0, if there is no word outside the parentheses),
* the number of words inside the parentheses (print 0, if there is no word inside the parentheses).

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 255) — the length of the given string. The second line contains the string consisting of only lowercase and uppercase English letters, parentheses and underscore symbols.

**Output**

Print two space-separated integers:

* the length of the longest word outside the parentheses (print 0, if there is no word outside the parentheses),
* the number of words inside the parentheses (print 0, if there is no word inside the parentheses).

**Examples**

**input**

37  
\_Hello\_Vasya(and\_Petya)\_\_bye\_(and\_OK)

**output**

5 4

**input**

37  
\_a\_(\_b\_\_\_c)\_\_de\_f(g\_)\_\_h\_\_i(j\_k\_l)m\_\_

**output**

2 6

**input**

27  
(LoooonG)\_\_shOrt\_\_(LoooonG)

**output**

5 2

**input**

5  
(\_\_\_)

**output**

0 0

**Note**

In the first sample, the words "Hello", "Vasya" and "bye" are outside any of the parentheses, and the words "and", "Petya", "and" and "OK" are inside. Note, that the word "and" is given twice and you should count it twice in the answer.

A. The New Year: Meeting Friends

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

There are three friend living on the straight line *Ox* in Lineland. The first friend lives at the point *x*1, the second friend lives at the point *x*2, and the third friend lives at the point *x*3. They plan to celebrate the New Year together, so they need to meet at one point. What is the minimum total distance they have to travel in order to meet at some point and celebrate the New Year?

It's guaranteed that the optimal answer is always integer.

**Input**

The first line of the input contains three **distinct** integers *x*1, *x*2 and *x*3 (1 ≤ *x*1, *x*2, *x*3 ≤ 100) — the coordinates of the houses of the first, the second and the third friends respectively.

**Output**

Print one integer — the minimum total distance the friends need to travel in order to meet together.

**Examples**

**input**

7 1 4

**output**

6

**input**

30 20 10

**output**

20

**Note**

In the first sample, friends should meet at the point 4. Thus, the first friend has to travel the distance of 3 (from the point 7 to the point 4), the second friend also has to travel the distance of 3 (from the point 1 to the point 4), while the third friend should not go anywhere because he lives at the point 4.

F. Cyclic Cipher

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given *n* sequences. Each sequence consists of positive integers, not exceeding *m*. All integers in one sequence are distinct, but the same integer may appear in multiple sequences. The length of the *i*-th sequence is *ki*.

Each second integers in each of the sequences are shifted by one to the left, i.e. integers at positions *i* > 1 go to positions *i* - 1, while the first integers becomes the last.

Each second we take the first integer of each sequence and write it down to a new array. Then, for each value *x* from 1 to *m* we compute the longest **segment** of the array consisting of element *x* only.

The above operation is performed for 10100 seconds. For each integer from 1 to *m* find out the longest segment found at this time.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 100 000) — the number of sequences and the maximum integer that can appear in the sequences.

Then follow *n* lines providing the sequences. Each of them starts with an integer *ki* (1 ≤ *ki* ≤ 40) — the number of integers in the sequence, proceeded by *ki* positive integers — elements of the sequence. It's guaranteed that all integers in each sequence are pairwise distinct and do not exceed *m*.

**The total length** of all sequences doesn't exceed 200 000.

**Output**

Print *m* integers, the *i*-th of them should be equal to the length of the longest segment of the array with all its values equal to *i* during the first 10100 seconds.

**Examples**

**input**

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

**output**

2  
1  
3  
2

**input**

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

**output**

3  
1  
4  
0  
1

**input**

4 6  
3 4 5 3  
2 6 3  
2 3 6  
3 3 6 5

**output**

0  
0  
2  
1  
1  
2

E. Research Rover

time limit per test

2.5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

*Unfortunately, the formal description of the task turned out to be too long, so here is the legend.*

Research rover finally reached the surface of Mars and is ready to complete its mission. Unfortunately, due to the mistake in the navigation system design, the rover is located in the wrong place.

The rover will operate on the grid consisting of *n* rows and *m* columns. We will define as (*r*, *c*) the cell located in the row *r* and column *c*. From each cell the rover is able to move to any cell that share a side with the current one.

The rover is currently located at cell (1, 1) and has to move to the cell (*n*, *m*). It will randomly follow some **shortest path** between these two cells. Each possible way is chosen equiprobably.

The cargo section of the rover contains the battery required to conduct the research. Initially, the battery charge is equal to *s* units of energy.

Some of the cells contain anomaly. Each time the rover gets to the cell with anomaly, the battery looses half of its charge rounded down. Formally, if the charge was equal to *x* before the rover gets to the cell with anomaly, the charge will change to http://codeforces.com/predownloaded/98/ea/98ea5b1147f0cd47f76c0c7a19aed60be7155d6f.png.

While the rover picks a random shortest path to proceed, compute the expected value of the battery charge after it reaches cell (*n*, *m*). If the cells (1, 1) and (*n*, *m*) contain anomaly, they also affect the charge of the battery.

**Input**

The first line of the input contains four integers *n*, *m*, *k* and *s* (1 ≤ *n*, *m* ≤ 100 000, 0 ≤ *k* ≤ 2000, 1 ≤ *s* ≤ 1 000 000) — the number of rows and columns of the field, the number of cells with anomaly and the initial charge of the battery respectively.

The follow *k* lines containing two integers *ri* and *ci* (1 ≤ *ri* ≤ *n*, 1 ≤ *ci* ≤ *m*) — coordinates of the cells, containing anomaly. It's guaranteed that each cell appears in this list no more than once.

**Output**

The answer can always be represented as an irreducible fraction http://codeforces.com/predownloaded/1d/99/1d99bb55692869d248f3e0a82cd378cfc79e13ec.png. Print the only integer *P*·*Q*- 1 modulo 109 + 7.

**Examples**

**input**

3 3 2 11  
2 1  
2 3

**output**

333333342

**input**

4 5 3 17  
1 2  
3 3  
4 1

**output**

514285727

**input**

1 6 2 15  
1 1  
1 5

**output**

4

**Note**

In the first sample, the rover picks one of the following six routes:

1. http://codeforces.com/predownloaded/17/e3/17e355eba0c4f50011e2aaf46c8139ab9941d4d7.png, after passing cell (2, 3) charge is equal to 6.
2. http://codeforces.com/predownloaded/4b/b6/4bb68cf5065033188acc3290cb1eab530a2de8b1.png, after passing cell (2, 3) charge is equal to 6.
3. http://codeforces.com/predownloaded/75/ae/75ae5f2937f1a5f2eb2fd66bd877de0125bb66d6.png, charge remains unchanged and equals 11.
4. http://codeforces.com/predownloaded/4f/cb/4fcbb6fb3b7630f3e0de94cb9343188e3a5b3f3b.png, after passing cells (2, 1) and (2, 3) charge equals 6 and then 3.
5. http://codeforces.com/predownloaded/2a/a4/2aa4fab52ed138f54cd2241c31b9817e9ed621e2.png, after passing cell (2, 1) charge is equal to 6.
6. http://codeforces.com/predownloaded/6c/82/6c827d2d90c5a43350c1176d6ec3fab211ea5a61.png, after passing cell (2, 1) charge is equal to 6.

Expected value of the battery charge is calculated by the following formula:

http://codeforces.com/predownloaded/d0/03/d0038e5df4118f19113ac2c11b37a59d638b8d25.png.

Thus *P* = 19, and *Q* = 3.

3- 1 modulo 109 + 7 equals 333333336.

19·333333336 = 333333342 (*mod* 109 + 7)

D. Generating Sets

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a set *Y* of *n* **distinct** positive integers *y*1, *y*2, ..., *yn*.

Set *X* of *n* **distinct** positive integers *x*1, *x*2, ..., *xn* is said to *generate* set *Y* if one can transform *X* to *Y* by applying some number of the following two operation to integers in *X*:

1. Take any integer *xi* and multiply it by two, i.e. replace *xi* with 2·*xi*.
2. Take any integer *xi*, multiply it by two and add one, i.e. replace *xi* with 2·*xi* + 1.

Note that integers in *X* are not required to be distinct after each operation.

Two sets of distinct integers *X* and *Y* are equal if they are equal as sets. In other words, if we write elements of the sets in the array in the increasing order, these arrays would be equal.

Note, that any set of integers (or its permutation) generates itself.

You are given a set *Y* and have to find a set *X* that generates *Y* and the **maximum element of *X* is mininum possible**.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 50 000) — the number of elements in *Y*.

The second line contains *n* integers *y*1, ..., *yn* (1 ≤ *yi* ≤ 109), that are guaranteed to be distinct.

**Output**

Print *n* integers — set of distinct integers that generate *Y* and the maximum element of which is minimum possible. If there are several such sets, print any of them.

**Examples**

**input**

5  
1 2 3 4 5

**output**

4 5 2 3 1

**input**

6  
15 14 3 13 1 12

**output**

12 13 14 7 3 1

**input**

6  
9 7 13 17 5 11

**output**

4 5 2 6 3 1

C. Destroying Array

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given an array consisting of *n* non-negative integers *a*1, *a*2, ..., *an*.

You are going to destroy integers in the array one by one. Thus, you are given the permutation of integers from 1 to *n* defining the order elements of the array are destroyed.

After each element is destroyed you have to find out the segment of the array, such that it contains no destroyed elements and the sum of its elements is maximum possible. The sum of elements in the empty segment is considered to be 0.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the length of the array.

The second line contains *n* integers *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 109).

The third line contains a permutation of integers from 1 to *n* — the order used to destroy elements.

**Output**

Print *n* lines. The *i*-th line should contain a single integer — the maximum possible sum of elements on the segment containing no destroyed elements, after first *i* operations are performed.

**Examples**

**input**

4  
1 3 2 5  
3 4 1 2

**output**

5  
4  
3  
0

**input**

5  
1 2 3 4 5  
4 2 3 5 1

**output**

6  
5  
5  
1  
0

**input**

8  
5 5 4 4 6 6 5 5  
5 2 8 7 1 3 4 6

**output**

18  
16  
11  
8  
8  
6  
6  
0

**Note**

Consider the first sample:

1. Third element is destroyed. Array is now 1 3  \*  5. Segment with maximum sum 5 consists of one integer 5.
2. Fourth element is destroyed. Array is now 1 3  \*   \* . Segment with maximum sum 4 consists of two integers 1 3.
3. First element is destroyed. Array is now  \*  3  \*   \* . Segment with maximum sum 3 consists of one integer 3.
4. Last element is destroyed. At this moment there are no valid nonempty segments left in this array, so the answer is equal to 0.

B. Verse Pattern

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a text consisting of *n* lines. Each line contains some space-separated words, consisting of lowercase English letters.

We define a syllable as a string that contains exactly one vowel and any arbitrary number (possibly none) of consonants. In English alphabet following letters are considered to be vowels: 'a', 'e', 'i', 'o', 'u' and 'y'.

Each word of the text that contains at least one vowel can be divided into syllables. Each character should be a part of exactly one syllable. For example, the word "mamma" can be divided into syllables as "ma" and "mma", "mam" and "ma", and "mamm" and "a". Words that consist of only consonants should be ignored.

The verse patterns for the given text is a sequence of *n* integers *p*1, *p*2, ..., *pn*. Text matches the given verse pattern if for each *i* from 1to *n* one can divide words of the *i*-th line in syllables in such a way that the total number of syllables is equal to *pi*.

You are given the text and the verse pattern. Check, if the given text matches the given verse pattern.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100) — the number of lines in the text.

The second line contains integers *p*1, ..., *pn* (0 ≤ *pi* ≤ 100) — the verse pattern.

Next *n* lines contain the text itself. Text consists of lowercase English letters and spaces. It's guaranteed that all lines are non-empty, each line starts and ends with a letter and words are separated by exactly one space. The length of each line doesn't exceed 100characters.

**Output**

If the given text matches the given verse pattern, then print "YES" (without quotes) in the only line of the output. Otherwise, print "NO" (without quotes).

**Examples**

**input**

3  
2 2 3  
intel  
code  
ch allenge

**output**

YES

**input**

4  
1 2 3 1  
a  
bcdefghi  
jklmnopqrstu  
vwxyz

**output**

NO

**input**

4  
13 11 15 15  
to be or not to be that is the question  
whether tis nobler in the mind to suffer  
the slings and arrows of outrageous fortune  
or to take arms against a sea of troubles

**output**

YES

**Note**

In the first sample, one can split words into syllables in the following way:

in-tel  
co-de  
ch al-len-ge

Since the word "ch" in the third line doesn't contain vowels, we can ignore it. As the result we get 2 syllabels in first two lines and 3syllables in the third one.

A. Broken Clock

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a broken clock. You know, that it is supposed to show time in 12- or 24-hours HH:MM format. In 12-hours format hours change from 1 to 12, while in 24-hours it changes from 0 to 23. In both formats minutes change from 0 to 59.

You are given a time in format HH:MM that is currently displayed on the broken clock. Your goal is to change minimum number of digits in order to make clocks display the correct time in the given format.

For example, if 00:99 is displayed, it is enough to replace the second 9 with 3 in order to get 00:39 that is a correct time in 24-hours format. However, to make 00:99 correct in 12-hours format, one has to change at least two digits. Additionally to the first change one can replace the second 0 with 1 and obtain 01:39.

**Input**

The first line of the input contains one integer 12 or 24, that denote 12-hours or 24-hours format respectively.

The second line contains the time in format HH:MM, that is currently displayed on the clock. First two characters stand for the hours, while next two show the minutes.

**Output**

The only line of the output should contain the time in format HH:MM that is a correct time in the given format. It should differ from the original in as few positions as possible. If there are many optimal solutions you can print any of them.

**Examples**

**input**

24  
17:30

**output**

17:30

**input**

12  
17:30

**output**

07:30

**input**

24  
99:99

**output**

09:09

E. Road to Home

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

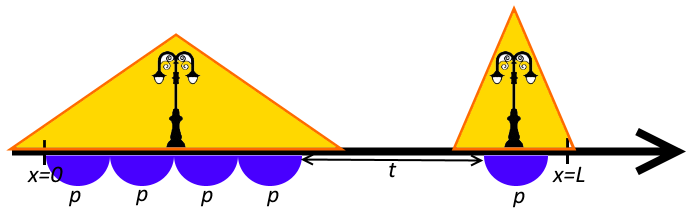
Once Danil the student was returning home from tram stop lately by straight road of length *L*. The stop is located at the point *x* = 0, but the Danil's home — at the point *x* = *L*. Danil goes from *x* = 0 to *x* = *L* with a constant speed and does not change direction of movement.

There are *n* street lights at the road, each of which lights some continuous segment of the road. All of the *n* lightened segments do not share common points.

Danil loves to sing, thus he wants to sing his favourite song over and over again during his walk. As soon as non-lightened segments of the road scare him, he sings only when he goes through the lightened segments.

Danil passes distance *p* while performing his favourite song once. Danil can't start another performance if the segment passed while performing is not fully lightened. Moreover, if Danil has taken a pause between two performances, he is not performing while not having passed a segment of length at least *t*. Formally,

1. Danil can start single performance at a point *x* only if every point of segment [*x*, *x* + *p*] is lightened;
2. If Danil has finished performing at a point *x* + *p*, then the next performance can be started only at a point *y* such that *y* = *x* + *p* or *y* ≥ *x* + *p* + *t* satisfying the statement under the point 1.

Blue half-circles denote performances. Please note that just after Danil has taken a pause in performing, he has not sang for a path of length of at least *t*.

Determine how many times Danil can perform his favourite song during his walk from *x* = 0 to *x* = *L*.

Please note that Danil does not break a single performance, thus, started singing another time, he finishes singing when having a segment of length of *p* passed from the performance start point.

**Input**

The first line of the input contains four integers *L*, *n*, *p* and *t* (1 ≤ *L* ≤ 109, 0 ≤ *n* ≤ 100 000, 1 ≤ *p* ≤ 109, 1 ≤ *t* ≤ 109) — the length of the Danil's path, the number of street lights at the road, the distance Danil passes while doing single performance and the minimum distance of pause respectively.

The next *n* lines describe segments lightened by street lights. *i*-th of them contains two integers *li*, *ri* (0 ≤ *li* < *ri* ≤ *L*) — the endpoints of the segment lightened by *i*-th street light. It is guaranteed that no two segments are intersecting, nesting, or touching each other. The segments are given in the order from left to right.

**Output**

Print the only integer — the maximum number of performances of Danil's favourite song on the path from *x* = 0 to *x* = *L*.

**Examples**

**input**

17 2 2 6  
0 9  
13 17

**output**

5

**input**

12 2 2 2  
0 5  
6 11

**output**

4

**input**

12 2 2 4  
0 5  
6 11

**output**

3

**Note**

The first sample case is just about corresponding to the picture from the statement.

D. Maxim and Array

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Recently Maxim has found an array of *n* integers, needed by no one. He immediately come up with idea of changing it: he invented positive integer *x* and decided to add or subtract it from arbitrary array elements. Formally, by applying single operation Maxim chooses integer *i* (1 ≤ *i* ≤ *n*) and replaces the *i*-th element of array *ai* either with *ai* + *x* or with *ai* - *x*. Please note that the operation may be applied more than once to the same position.

Maxim is a curious minimalis, thus he wants to know what is the minimum value that the product of all array elements (i.e. http://codeforces.com/predownloaded/4d/5a/4d5ae9233afc8f3ab70c1a3f505413c96f0e1b8b.png) can reach, if Maxim would apply no more than *k* operations to it. Please help him in that.

**Input**

The first line of the input contains three integers *n*, *k* and *x* (1 ≤ *n*, *k* ≤ 200 000, 1 ≤ *x* ≤ 109) — the number of elements in the array, the maximum number of operations and the number invented by Maxim, respectively.

The second line contains *n* integers *a*1, *a*2, ..., *an* (http://codeforces.com/predownloaded/99/f0/99f0b23d9ccd46839a6d9f9821172bc3db4b1a60.png) — the elements of the array found by Maxim.

**Output**

Print *n* integers *b*1, *b*2, ..., *bn* in the only line — the array elements after applying no more than *k* operations to the array. In particular, http://codeforces.com/predownloaded/83/1c/831cc5bea4cea6aa07e411a95acaa76959eecd76.png should stay true for every 1 ≤ *i* ≤ *n*, but the product of all array elements should be **minimum possible**.

If there are multiple answers, print any of them.

**Examples**

**input**

5 3 1  
5 4 3 5 2

**output**

5 4 3 5 -1

**input**

5 3 1  
5 4 3 5 5

**output**

5 4 0 5 5

**input**

5 3 1  
5 4 4 5 5

**output**

5 1 4 5 5

**input**

3 2 7  
5 4 2

**output**

5 11 -5

C. Journey

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Recently Irina arrived to one of the most famous cities of Berland — the Berlatov city. There are *n* showplaces in the city, numbered from 1 to *n*, and some of them are connected by one-directional roads. The roads in Berlatov are designed in a way such that there **are no** cyclic routes between showplaces.

Initially Irina stands at the showplace 1, and the endpoint of her journey is the showplace *n*. Naturally, Irina wants to visit as much showplaces as she can during her journey. However, Irina's stay in Berlatov is limited and she can't be there for more than *T* time units.

Help Irina determine how many showplaces she may visit during her journey from showplace 1 to showplace *n* within a time not exceeding *T*. It is guaranteed that there is at least one route from showplace 1 to showplace *n* such that Irina will spend no more than *T*time units passing it.

**Input**

The first line of the input contains three integers *n*, *m* and *T* (2 ≤ *n* ≤ 5000,  1 ≤ *m* ≤ 5000,  1 ≤ *T* ≤ 109) — the number of showplaces, the number of roads between them and the time of Irina's stay in Berlatov respectively.

The next *m* lines describes roads in Berlatov. *i*-th of them contains 3 integers *ui*, *vi*, *ti* (1 ≤ *ui*, *vi* ≤ *n*, *ui* ≠ *vi*, 1 ≤ *ti* ≤ 109), meaning that there is a road starting from showplace *ui* and leading to showplace *vi*, and Irina spends *ti* time units to pass it. It is guaranteed that the roads do not form cyclic routes.

**It is guaranteed, that there is at most one road between each pair of showplaces.**

**Output**

Print the single integer *k* (2 ≤ *k* ≤ *n*) — the maximum number of showplaces that Irina can visit during her journey from showplace 1 to showplace *n* within time not exceeding *T*, in the first line.

Print *k* distinct integers in the second line — indices of showplaces that Irina will visit on her route, in the order of encountering them.

If there are multiple answers, print any of them.

**Examples**

**input**

4 3 13  
1 2 5  
2 3 7  
2 4 8

**output**

3  
1 2 4

**input**

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

**output**

4  
1 2 4 6

**input**

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

**output**

3  
1 3 5

B. Passwords

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Vanya is managed to enter his favourite site Codehorses. Vanya uses *n* distinct passwords for sites at all, however he can't remember which one exactly he specified during Codehorses registration.

Vanya will enter passwords in order of non-decreasing their lengths, and he will enter passwords of same length in arbitrary order. Just when Vanya will have entered the correct password, he is immediately authorized on the site. Vanya will not enter any password twice.

Entering any passwords takes one second for Vanya. But if Vanya will enter wrong password *k* times, then he is able to make the next try only 5 seconds after that. Vanya makes each try immediately, that is, at each moment when Vanya is able to enter password, he is doing that.

Determine how many seconds will Vanya need to enter Codehorses in the best case for him (if he spends minimum possible number of second) and in the worst case (if he spends maximum possible amount of seconds).

**Input**

The first line of the input contains two integers *n* and *k* (1 ≤ *n*, *k* ≤ 100) — the number of Vanya's passwords and the number of failed tries, after which the access to the site is blocked for 5 seconds.

The next *n* lines contains passwords, one per line — pairwise distinct non-empty strings consisting of latin letters and digits. Each password length does not exceed 100 characters.

The last line of the input contains the Vanya's Codehorses password. It is guaranteed that the Vanya's Codehorses password is equal to some of his *n* passwords.

**Output**

Print two integers — time (in seconds), Vanya needs to be authorized to Codehorses in the best case for him and in the worst case respectively.

**Examples**

**input**

5 2  
cba  
abc  
bb1  
abC  
ABC  
abc

**output**

1 15

**input**

4 100  
11  
22  
1  
2  
22

**output**

3 4

**Note**

Consider the first sample case. As soon as all passwords have the same length, Vanya can enter the right password at the first try as well as at the last try. If he enters it at the first try, he spends exactly 1 second. Thus in the best case the answer is 1. If, at the other hand, he enters it at the last try, he enters another 4 passwords before. He spends 2 seconds to enter first 2 passwords, then he waits 5seconds as soon as he made 2 wrong tries. Then he spends 2 more seconds to enter 2 wrong passwords, again waits 5 seconds and, finally, enters the correct password spending 1 more second. In summary in the worst case he is able to be authorized in 15 seconds.

Consider the second sample case. There is no way of entering passwords and get the access to the site blocked. As soon as the required password has length of 2, Vanya enters all passwords of length 1 anyway, spending 2 seconds for that. Then, in the best case, he immediately enters the correct password and the answer for the best case is 3, but in the worst case he enters wrong password of length 2 and only then the right one, spending 4 seconds at all.

A. One-dimensional Japanese Crossword

time limit per test

1 second

memory limit per test

256 megabytes

input

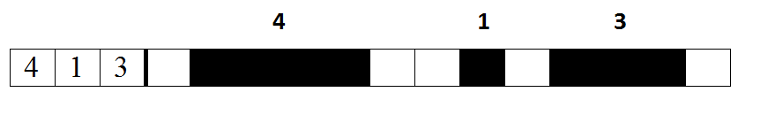
standard input

output

standard output

Recently Adaltik discovered japanese crosswords. Japanese crossword is a picture, represented as a table sized *a* × *b* squares, and each square is colored white or black. There are integers to the left of the rows and to the top of the columns, encrypting the corresponding row or column. The number of integers represents how many groups of black squares there are in corresponding row or column, and the integers themselves represents the number of consecutive black squares in corresponding group (you can find more detailed explanation in Wikipedia <https://en.wikipedia.org/wiki/Japanese_crossword>).

Adaltik decided that the general case of japanese crossword is too complicated and drew a row consisting of *n* squares (e.g. japanese crossword sized 1 × *n*), which he wants to encrypt in the same way as in japanese crossword.

The example of encrypting of a single row of japanese crossword.

Help Adaltik find the numbers encrypting the row he drew.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100) — the length of the row. The second line of the input contains a single string consisting of *n* characters 'B' or 'W', ('B' corresponds to black square, 'W' — to white square in the row that Adaltik drew).

**Output**

The first line should contain a single integer *k* — the number of integers encrypting the row, e.g. the number of groups of black squares in the row.

The second line should contain *k* integers, encrypting the row, e.g. corresponding to sizes of groups of consecutive black squares in the order from left to right.

**Examples**

**input**

3  
BBW

**output**

1  
2

**input**

5  
BWBWB

**output**

3  
1 1 1

**input**

4  
WWWW

**output**

0

**input**

4  
BBBB

**output**

1  
4

**input**

13  
WBBBBWWBWBBBW

**output**

3  
4 1 3

**Note**

The last sample case correspond to the picture in the statement.

F. Array Covering

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Misha has an array of integers of length *n*. He wants to choose *k* different continuous subarrays, so that each element of the array belongs to at least one of the chosen subarrays.

Misha wants to choose the subarrays in such a way that if he calculated the sum of elements for each subarray, and then add up all these sums, the resulting value was maximum possible.

**Input**

The first line of input contains two integers: *n*, *k* (1 ≤ *n* ≤ 100 000, 1 ≤ *k* ≤ *n*·(*n* + 1) / 2) — the number of elements in the array and the number of different subarrays that must be chosen.

The second line contains *n* integers *ai* ( - 50 000 ≤ *ai* ≤ 50 000) — the elements of the array.

**Output**

Output one integer — the maximum possible value Misha can get by choosing *k* different subarrays.

**Example**

**input**

5 4  
6 -4 -10 -4 7

**output**

11

E. Cipher

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Borya has recently found a big electronic display. The computer that manages the display stores some integer number. The number has *n* decimal digits, the display shows the encoded version of the number, where each digit is shown using some lowercase letter of the English alphabet.

There is a legend near the display, that describes how the number is encoded. For each digit position *i* and each digit *j* the character *c*is known, that encodes this digit at this position. Different digits can have the same code characters.

Each second the number is increased by 1. And one second after a moment when the number reaches the value that is represented as *n* 9-s in decimal notation, the loud beep sounds.

Andrew knows the number that is stored in the computer. Now he wants to know how many seconds must pass until Borya can definitely tell what was the original number encoded by the display. Assume that Borya can precisely measure time, and that the encoded number will first be increased exactly one second after Borya started watching at the display.

**Input**

Input data contains multiple test cases. The first line of input contains *t* (1 ≤ *t* ≤ 100) — the number of test cases.

Each test case is described as follows. The first line of the description contains *n* (1 ≤ *n* ≤ 18) — the number of digits in the number. The second line contains *n* decimal digits without spaces (but possibly with leading zeroes) — the number initially stored in the display computer. The following *n* lines contain 10 characters each. The *j*-th character of the *i*-th of these lines is the code character for a digit *j* - 1 in position *i*, most significant digit positions are described first.

**Output**

For each test case print an integer: the number of seconds until Borya definitely knows what was the initial number stored on the display of the computer. Do not print leading zeroes.

**Example**

**input**

3  
2  
42  
abcdefghij  
jihgfedcba  
2  
42  
aaaaaaaaaa  
aaaaaaaaaa  
1  
2  
abcdabcdff

**output**

0  
58  
2

D. Slalom

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

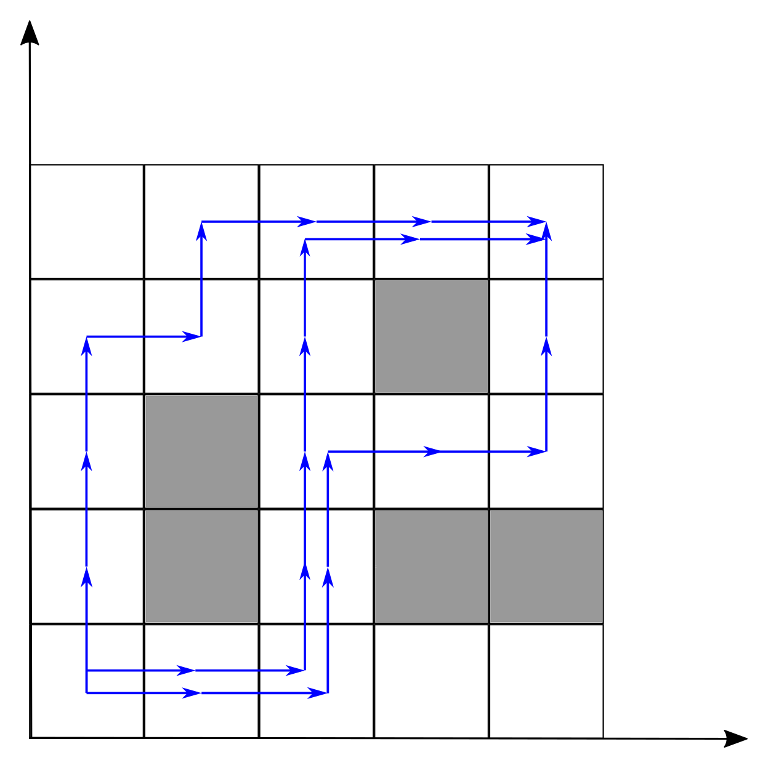
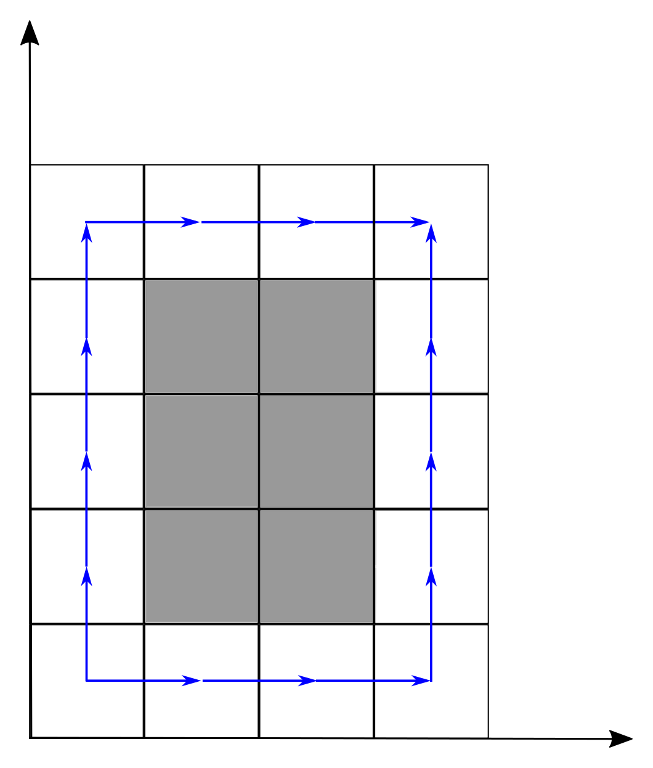
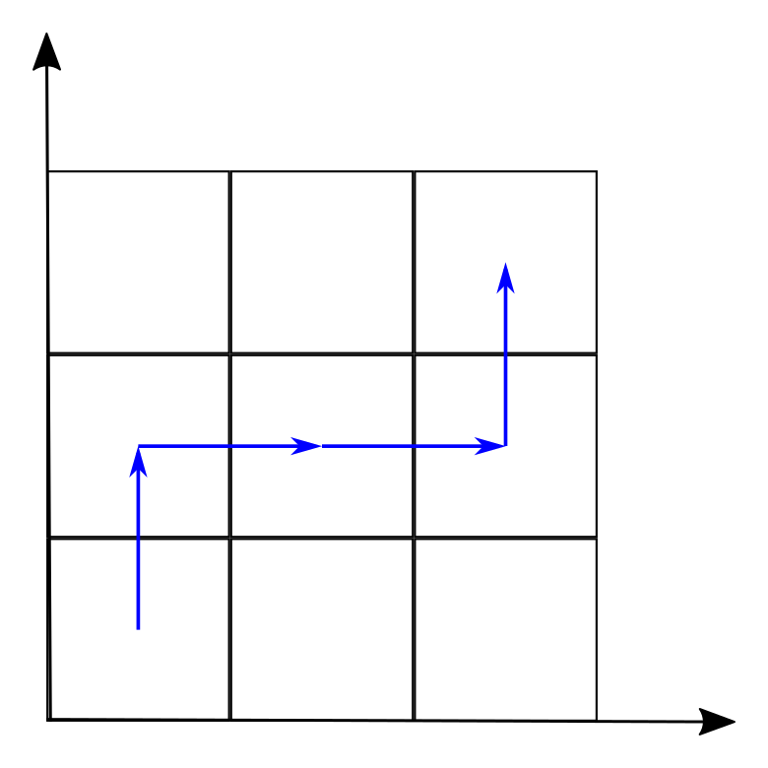
standard output

Little girl Masha likes winter sports, today she's planning to take part in slalom skiing.

The track is represented as a grid composed of *n* × *m* squares. There are rectangular obstacles at the track, composed of grid squares. Masha must get from the square (1, 1) to the square (*n*, *m*). She can move from a square to adjacent square: either to the right, or upwards. If the square is occupied by an obstacle, it is not allowed to move to that square.

One can see that each obstacle can actually be passed in two ways: either it is to the right of Masha's path, or to the left. Masha likes to try all ways to do things, so she would like to know how many ways are there to pass the track. Two ways are considered different if there is an obstacle such that it is to the right of the path in one way, and to the left of the path in the other way.

Help Masha to find the number of ways to pass the track. The number of ways can be quite big, so Masha would like to know it modulo 109 + 7.

The pictures below show different ways to pass the track in sample tests.

**Input**

The first line of input data contains three positive integers: *n*, *m* and *k* (3 ≤ *n*, *m* ≤ 106, 0 ≤ *k* ≤ 105) — the size of the track and the number of obstacles.

The following *k* lines contain four positive integers each: *x*1, *y*1, *x*2, *y*2 (1 ≤ *x*1 ≤ *x*2 ≤ *n*, 1 ≤ *y*1 ≤ *y*2 ≤ *m*) — coordinates of bottom left, and top right squares of the obstacle.

It is guaranteed that there are no obstacles at squares (1, 1) and (*n*, *m*), and no obstacles overlap (but some of them may touch).

**Output**

Output one integer — the number of ways to pass the track modulo 109 + 7.

**Examples**

**input**

3 3 0

**output**

1

**input**

4 5 1  
2 2 3 4

**output**

2

**input**

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

**output**

3

C. Homework

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Today Peter has got an additional homework for tomorrow. The teacher has given three integers to him: *n*, *m* and *k*, and asked him to mark one or more squares on a square grid of size *n* × *m*.

The marked squares must form a connected figure, and there must be exactly *k* triples of marked squares that form an L-shaped tromino — all three squares are inside a 2 × 2 square.

The set of squares forms a connected figure if it is possible to get from any square to any other one if you are allowed to move from a square to any adjacent by a common side square.

Peter cannot fulfill the task, so he asks you for help. Help him to create such figure.

**Input**

Input data contains one or more test cases. The first line contains the number of test cases *t* (1 ≤ *t* ≤ 100).

Each of the following *t* test cases is described by a line that contains three integers: *n*, *m* and *k* (3 ≤ *n*, *m*, *n* × *m* ≤ 105, 0 ≤ *k* ≤ 109).

The sum of values of *n* × *m* for all tests in one input data doesn't exceed 105.

**Output**

For each test case print the answer.

If it is possible to create such figure, print *n* lines, *m* characters each, use asterisk '\*' to denote the marked square, and dot '.' to denote the unmarked one.

If there is no solution, print -1.

Print empty line between test cases.

**Example**

**input**

3  
3 3 4  
3 3 5  
3 3 3

**output**

.\*.  
\*\*\*  
.\*.  
  
\*\*.  
\*\*.  
\*..  
  
.\*.  
\*\*\*  
\*..

B. Cactusophobia

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Tree is a connected undirected graph that has no cycles. Edge cactus is a connected undirected graph without loops and parallel edges, such that each edge belongs to at most one cycle.

Vasya has an edge cactus, each edge of this graph has some color.

Vasya would like to remove the minimal number of edges in such way that his cactus turned to a tree. Vasya wants to make it in such a way that there were edges of as many different colors in the resulting tree, as possible. Help him to find how many different colors can the resulting tree have.

**Input**

The first line contains two integers: *n*, *m* (2 ≤ *n* ≤ 10 000) — the number of vertices and the number of edges in Vasya's graph, respectively.

The following *m* lines contain three integers each: *u*, *v*, *c* (1 ≤ *u*, *v* ≤ *n*, *u* ≠ *v*, 1 ≤ *c* ≤ *m*) — the numbers of vertices connected by the corresponding edge, and its color. It is guaranteed that the described graph is indeed an edge cactus.

**Output**

Output one integer: the maximal number of different colors that the resulting tree can have.

**Examples**

**input**

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

**output**

3

**input**

7 9  
1 2 1  
2 3 4  
3 1 5  
1 4 5  
4 5 2  
5 1 6  
1 6 4  
6 7 6  
7 1 3

**output**

6

A. Closing ceremony

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

The closing ceremony of Squanch Code Cup is held in the big hall with *n* × *m* seats, arranged in *n* rows, *m* seats in a row. Each seat has two coordinates (*x*, *y*) (1 ≤ *x* ≤ *n*, 1 ≤ *y* ≤ *m*).

There are two queues of people waiting to enter the hall: *k* people are standing at (0, 0) and *n*·*m* - *k* people are standing at (0, *m* + 1). Each person should have a ticket for a specific seat. If person *p* at (*x*, *y*) has ticket for seat (*xp*, *yp*) then he should walk |*x* - *xp*| + |*y* - *yp*|to get to his seat.

Each person has a stamina — the maximum distance, that the person agrees to walk. You should find out if this is possible to distribute all *n*·*m* tickets in such a way that each person has enough stamina to get to their seat.

**Input**

The first line of input contains two integers *n* and *m* (1 ≤ *n*·*m* ≤ 104) — the size of the hall.

The second line contains several integers. The first integer *k* (0 ≤ *k* ≤ *n*·*m*) — the number of people at (0, 0). The following *k* integers indicate stamina of each person there.

The third line also contains several integers. The first integer *l* (*l* = *n*·*m* - *k*) — the number of people at (0, *m* + 1). The following *l*integers indicate stamina of each person there.

The stamina of the person is a positive integer less that or equal to *n* + *m*.

**Output**

If it is possible to distribute tickets between people in the described manner print "YES", otherwise print "NO".

**Examples**

**input**

2 2  
3 3 3 2  
1 3

**output**

YES

**input**

2 2  
3 2 3 3  
1 2

**output**

NO

B. Anatoly and Cockroaches

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Anatoly lives in the university dorm as many other students do. As you know, cockroaches are also living there together with students. Cockroaches might be of two colors: black and red. There are *n* cockroaches living in Anatoly's room.

Anatoly just made all his cockroaches to form a single line. As he is a perfectionist, he would like the colors of cockroaches in the line to **alternate**. He has a can of black paint and a can of red paint. In one turn he can either swap any two cockroaches, or take any single cockroach and change it's color.

Help Anatoly find out the minimum number of turns he needs to make the colors of cockroaches in the line alternate.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the number of cockroaches.

The second line contains a string of length *n*, consisting of characters 'b' and 'r' that denote black cockroach and red cockroach respectively.

**Output**

Print one integer — the minimum number of moves Anatoly has to perform in order to make the colors of cockroaches in the line to alternate.

**Examples**

**input**

5  
rbbrr

**output**

1

**input**

5  
bbbbb

**output**

2

**input**

3  
rbr

**output**

0

**Note**

In the first sample, Anatoly has to swap third and fourth cockroaches. He needs 1 turn to do this.

In the second sample, the optimum answer is to paint the second and the fourth cockroaches red. This requires 2 turns.

In the third sample, the colors of cockroaches in the line are alternating already, thus the answer is 0.

A. Vitya in the Countryside

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Every summer Vitya comes to visit his grandmother in the countryside. This summer, he got a huge wart. Every grandma knows that one should treat warts when the moon goes down. Thus, Vitya has to catch the moment when the moon is down.

Moon cycle lasts 30 days. The size of the visible part of the moon (in Vitya's units) for each day is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, and then cycle repeats, thus after the second 1 again goes 0.

As there is no internet in the countryside, Vitya has been watching the moon for *n* consecutive days and for each of these days he wrote down the size of the visible part of the moon. Help him find out whether the moon will be up or down next day, or this cannot be determined by the data he has.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 92) — the number of consecutive days Vitya was watching the size of the visible part of the moon.

The second line contains *n* integers *ai* (0 ≤ *ai* ≤ 15) — Vitya's records.

It's guaranteed that the input data is consistent.

**Output**

If Vitya can be sure that the size of visible part of the moon on day *n* + 1 will be less than the size of the visible part on day *n*, then print "DOWN" at the only line of the output. If he might be sure that the size of the visible part will increase, then print "UP". If it's impossible to determine what exactly will happen with the moon, print -1.

**Examples**

**input**

5  
3 4 5 6 7

**output**

UP

**input**

7  
12 13 14 15 14 13 12

**output**

DOWN

**input**

1  
8

**output**

-1

**Note**

In the first sample, the size of the moon on the next day will be equal to 8, thus the answer is "UP".

In the second sample, the size of the moon on the next day will be 11, thus the answer is "DOWN".

In the third sample, there is no way to determine whether the size of the moon on the next day will be 7 or 9, thus the answer is -1.

E. Matvey's Birthday

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Today is Matvey's birthday. He never knows what to ask as a present so friends gave him a string *s* of length *n*. This string consists of only first eight English letters: 'a', 'b', ..., 'h'.

First question that comes to mind is: who might ever need some string? Matvey is a special boy so he instantly found what to do with this string. He used it to build an undirected graph where vertices correspond to position in the string and there is an edge between distinct positions *a* and *b* (1 ≤ *a*, *b* ≤ *n*) if **at least one** of the following conditions hold:

1. *a* and *b* are neighbouring, i.e. |*a* - *b*| = 1.
2. Positions *a* and *b* contain equal characters, i.e. *sa* = *sb*.

Then Matvey decided to find the diameter of this graph. Diameter is a maximum distance (length of the shortest path) among all pairs of vertices. Also, Matvey wants to find the number of pairs of vertices such that the distance between them is equal to the diameter of the graph. As he is very cool and experienced programmer he managed to solve this problem very fast. Will you do the same?

**Input**

The first line of the input contains a single integer *n* (2 ≤ *n* ≤ 100 000) — the length of the string.

The second line contains the string *s* itself. It's guaranteed that *s* consists of only first eight letters of English alphabet.

**Output**

Print two integers — the diameter of the graph and the number of pairs of positions with the distance equal to the diameter.

**Examples**

**input**

3  
abc

**output**

2 1

**input**

7  
aaabaaa

**output**

2 4

**Note**

Consider the second sample.

The maximum distance is 2. It's obtained for pairs (1, 4), (2, 4), (4, 6) and (4, 7).

D. Andrew and Chemistry

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

*During the chemistry lesson Andrew learned that the saturated hydrocarbons (alkanes) enter into radical chlorination reaction. Andrew is a very curious boy, so he wondered how many different products of the reaction may be forms for a given alkane. He managed to solve the task for small molecules, but for large ones he faced some difficulties and asks you to help.*

Formally, you are given a tree consisting of *n* vertices, such that the degree of each vertex doesn't exceed 4. You have to count the number of distinct non-isomorphic trees that can be obtained by adding to this tree one new vertex and one new edge, such that the graph is still the tree and the degree of each vertex doesn't exceed 4.

Two trees are isomorphic if there exists a bijection *f*(*v*) such that vertices *u* and *v* are connected by an edge if and only if vertices *f*(*v*)and *f*(*u*) are connected by an edge.

**Input**

The first line of the input contains an integer *n* (1 ≤ *n* ≤ 100 000) — the number of vertices in the tree.

Then follow *n* - 1 lines with edges descriptions. Each edge is given by two integers *ui* and *vi* (1 ≤ *ui*, *vi* ≤ *n*) — indices of vertices connected by an edge. It's guaranteed that the given graph is a tree and the degree of each vertex doesn't exceed 4.

**Output**

Print one integer — the answer to the question.

**Examples**

**input**

4  
1 2  
2 3  
2 4

**output**

2

**input**

5  
1 2  
1 3  
1 4  
1 5

**output**

1

**input**

5  
2 5  
5 3  
4 3  
4 1

**output**

3

**Note**

In the first sample, one can add new vertex to any existing vertex, but the trees we obtain by adding a new vertex to vertices 1, 3 and 4are isomorphic, thus the answer is 2.

In the second sample, one can't add new vertex to the first vertex, as its degree is already equal to four. Trees, obtained by adding a new vertex to vertices 2, 3, 4 and 5 are isomorphic, thus the answer is 1.

C. Sasha and Array

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Sasha has an array of integers *a*1, *a*2, ..., *an*. You have to perform *m* queries. There might be queries of two types:

1. 1 l r x — increase all integers on the segment from *l* to *r* by values *x*;
2. 2 l r — find http://codeforces.com/predownloaded/21/60/21609daba8f7bc13a6b19ca8be77d0d65a503521.png, where *f*(*x*) is the *x*-th Fibonacci number. As this number may be large, you only have to find it modulo 109 + 7.

In this problem we define Fibonacci numbers as follows: *f*(1) = 1, *f*(2) = 1, *f*(*x*) = *f*(*x* - 1) + *f*(*x* - 2) for all *x* > 2.

Sasha is a very talented boy and he managed to perform all queries in five seconds. Will you be able to write the program that performs as well as Sasha?

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n* ≤ 100 000, 1 ≤ *m* ≤ 100 000) — the number of elements in the array and the number of queries respectively.

The next line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109).

Then follow *m* lines with queries descriptions. Each of them contains integers *tpi*, *li*, *ri* and may be *xi* (1 ≤ *tpi* ≤ 2, 1 ≤ *li* ≤ *ri* ≤ *n*, 1 ≤ *xi* ≤ 109). Here *tpi* = 1 corresponds to the queries of the first type and *tpi* corresponds to the queries of the second type.

It's guaranteed that the input will contains at least one query of the second type.

**Output**

For each query of the second type print the answer modulo 109 + 7.

**Examples**

**input**

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

**output**

5  
7  
9

**Note**

Initially, array *a* is equal to 1, 1, 2, 1, 1.

The answer for the first query of the second type is *f*(1) + *f*(1) + *f*(2) + *f*(1) + *f*(1) = 1 + 1 + 1 + 1 + 1 = 5.

After the query 1 2 4 2 array *a* is equal to 1, 3, 4, 3, 1.

The answer for the second query of the second type is *f*(3) + *f*(4) + *f*(3) = 2 + 3 + 2 = 7.

The answer for the third query of the second type is *f*(1) + *f*(3) + *f*(4) + *f*(3) + *f*(1) = 1 + 2 + 3 + 2 + 1 = 9.

A. Efim and Strange Grade

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Efim just received his grade for the last test. He studies in a special school and his grade can be equal to any positive decimal fraction. First he got disappointed, as he expected a way more pleasant result. Then, he developed a tricky plan. Each second, he can ask his teacher to round the grade at any place after the decimal point (also, he can ask to round to the nearest integer).

There are *t* seconds left till the end of the break, so Efim has to act fast. Help him find what is the maximum grade he can get in no more than *t* seconds. Note, that he can choose to not use all *t* seconds. Moreover, he can even choose to not round the grade at all.

In this problem, classic rounding rules are used: while rounding number to the *n*-th digit one has to take a look at the digit *n* + 1. If it is less than 5 than the *n*-th digit remain unchanged while all subsequent digits are replaced with 0. Otherwise, if the *n* + 1 digit is greater or equal to 5, the digit at the position *n* is increased by 1 (this might also change some other digits, if this one was equal to 9) and all subsequent digits are replaced with 0. At the end, all trailing zeroes are thrown away.

For example, if the number 1.14 is rounded to the first decimal place, the result is 1.1, while if we round 1.5 to the nearest integer, the result is 2. Rounding number 1.299996121 in the fifth decimal place will result in number 1.3.

**Input**

The first line of the input contains two integers *n* and *t* (1 ≤ *n* ≤ 200 000, 1 ≤ *t* ≤ 109) — the length of Efim's grade and the number of seconds till the end of the break respectively.

The second line contains the grade itself. It's guaranteed that the grade is a positive number, containing at least one digit after the decimal points, and it's representation doesn't finish with 0.

**Output**

Print the maximum grade that Efim can get in *t* seconds. Do not print trailing zeroes.

**Examples**

**input**

6 1  
10.245

**output**

10.25

**input**

6 2  
10.245

**output**

10.3

**input**

3 100  
9.2

**output**

9.2

**Note**

In the first two samples Efim initially has grade 10.245.

During the first second Efim can obtain grade 10.25, and then 10.3 during the next second. Note, that the answer 10.30 will be considered incorrect.

In the third sample the optimal strategy is to not perform any rounding at all.

I. Cowboy Beblop at his computer

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Cowboy Beblop is a funny little boy who likes sitting at his computer. He somehow obtained two elastic hoops in the shape of 2D polygons, which are not necessarily convex. Since there's no gravity on his spaceship, the hoops are standing still in the air. Since the hoops are very elastic, Cowboy Beblop can stretch, rotate, translate or shorten their edges as much as he wants.

For both hoops, you are given the number of their vertices, as well as the position of each vertex, defined by the X , Y and Z coordinates. The vertices are given in the order they're connected: the 1st vertex is connected to the 2nd, which is connected to the 3rd, etc., and the last vertex is connected to the first one. Two hoops are connected if it's impossible to pull them to infinity in different directions by manipulating their edges, without having their edges or vertices intersect at any point – **just like when two links of a chain are connected**. **The polygons' edges do not intersect or overlap**.

To make things easier, we say that two polygons are **well-connected**, if the edges of one polygon cross the area of the other polygon in two different directions (from the upper and lower sides of the plane defined by that polygon) a different number of times.

Cowboy Beblop is fascinated with the hoops he has obtained and he would like to know whether they are well-connected or not. Since he’s busy playing with his dog, Zwei, he’d like you to figure it out for him. He promised you some sweets if you help him!

**Input**

The first line of input contains an integer *n* (3 ≤ *n* ≤ 100 000), which denotes the number of edges of the first polygon. The next N lines each contain the integers *x*, *y* and *z* ( - 1 000 000 ≤ *x*, *y*, *z* ≤ 1 000 000) — coordinates of the vertices, in the manner mentioned above. The next line contains an integer *m* (3 ≤ *m* ≤ 100 000) , denoting the number of edges of the second polygon, followed by *m*lines containing the coordinates of the second polygon’s vertices.

It is guaranteed that both polygons are simple (no self-intersections), and in general that **the obtained polygonal lines do not intersect each other**. Also, you can assume that no 3 consecutive points of a polygon lie on the same line.

**Output**

Your output should contain only one line, with the words "YES" or "NO", depending on whether the two given polygons are well-connected.

**Example**

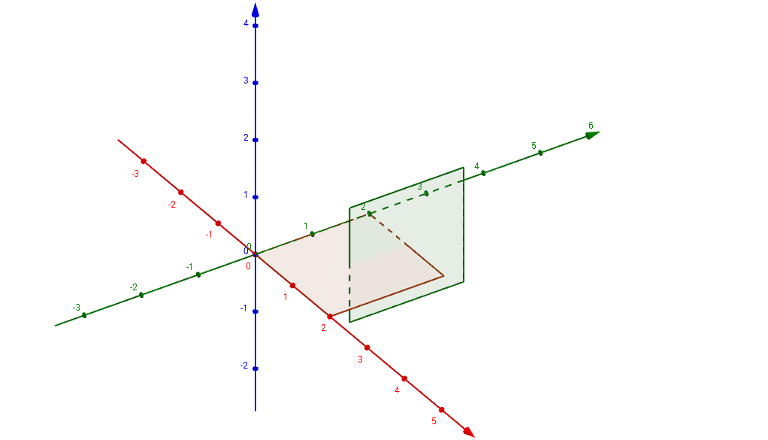
**input**

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

**output**

YES

**Note**

On the picture below, the two polygons are well-connected, as the edges of the vertical polygon cross the area of the horizontal one exactly once in one direction (for example, from above to below), and zero times in the other (in this case, from below to above). Note that the polygons do not have to be parallel to any of the xy-,xz-,yz- planes in general.

H. Pokermon League challenge

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Welcome to the world of Pokermon, yellow little mouse-like creatures, who absolutely love playing poker!

Yeah, right…

In the ensuing Pokermon League, there are *n* registered Pokermon trainers, and *t* existing trainer teams each of which belongs to one of two conferences. Since there is a lot of jealousy between trainers, there are *e* pairs of trainers who hate each other. Their hate is mutual, there are no identical pairs among these, and no trainer hates himself (the world of Pokermon is a joyful place!). Each trainer has a wish-list of length *li* of teams he’d like to join.

Your task is to divide players into teams and the teams into two conferences, so that:

* each trainer belongs to exactly one team;
* no team is in both conferences;
* total hate between conferences is at least *e* / 2;
* every trainer is in a team from his wish-list.

Total hate between conferences is calculated as the number of pairs of trainers from teams from different conferences who hate each other.

**Input**

The first line of the input contains two integer *n* (4 ≤ *n* ≤ 50 000) and *e* (2 ≤ *e* ≤ 100 000) — the total number of Pokermon trainers and the number of pairs of trainers who hate each other.

Pokermon trainers are numbered from 1 to *n*. Next *e* lines contain two integers *a* and *b* (1 ≤ *a*, *b* ≤ *n*) indicating that Pokermon trainers *a* and *b* hate each other. Next 2*n* lines are in a following format. Starting with Pokermon trainer 1, for each trainer in consecutive order: first number *li* (16 ≤ *li* ≤ 20) — a size of Pokermon trainers wish list, then *li* positive integers *ti*,*j* (1 ≤ *ti*,*j* ≤ *T*), providing the teams the *i*-th trainer would like to be on.

Each trainers wish list will contain each team no more than once. Teams on the wish lists are numbered in such a way that the set of all teams that appear on at least 1 wish list is set of consecutive positive integers {1, 2, 3, …, *T*}. Here *T* might be up to 1 000 000.

**Output**

Print two lines. The first line should contain *n* numbers, specifying for each trainer the team he is in.

The second line should contain *T* numbers, specifying the conference for each team (1 or 2).

**Example**

**input**

4 3  
1 2  
2 3  
4 1  
16  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 15  
16  
2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18  
16  
2 3 4 5 6 7 8 9 10 11 12 13 14 15 18 19  
16  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 19

**output**

16 15 19 14   
2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 1

**Note**

G. Underfail

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You have recently fallen through a hole and, after several hours of unconsciousness, have realized you are in an underground city. On one of your regular, daily walks through the unknown, you have encountered two unusually looking skeletons called Sanz and P’pairus, who decided to accompany you and give you some puzzles for seemingly unknown reasons.

One day, Sanz has created a crossword for you. Not any kind of crossword, but a 1D crossword! You are given *m* words and a string of length *n*. You are also given an array *p*, which designates how much each word is worth — the *i*-th word is worth *pi* points. Whenever you find one of the *m* words in the string, you are given the corresponding number of points. Each position in the crossword can be used at most *x* times. A certain word can be counted at different places, but you cannot count the same appearance of a word multiple times. If a word is a substring of another word, you can count them both (presuming you haven’t used the positions more than *x* times).

In order to solve the puzzle, you need to tell Sanz what’s the maximum achievable number of points in the crossword. There is no need to cover all postions, just get the maximal score! Crossword and words contain only lowercase English letters.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 500) — the length of the crossword. The second line contains the crossword string. The third line contains a single integer *m* (1 ≤ *m* ≤ 100) — the number of given words, and next *m* lines contain description of words: each line will have a string representing a non-empty word (its length doesn't exceed the length of the crossword) and integer *pi* (0 ≤ *pi* ≤ 100). Last line of the input will contain *x* (1 ≤ *x* ≤ 100) — maximum number of times a position in crossword can be used.

**Output**

Output single integer — maximum number of points you can get.

**Example**

**input**

6  
abacba  
2  
aba 6  
ba 3  
3

**output**

12

**Note**

For example, with the string "abacba", words "aba" (6 points) and "ba" (3 points), and *x* = 3, you can get at most 12 points - the word "aba" appears once ("abacba"), while "ba" appears two times ("abacba"). Note that for *x* = 1, you could get at most 9 points, since you wouldn’t be able to count both "aba" and the first appearance of "ba".

F. Heroes of Making Magic III

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

I’m strolling on sunshine, yeah-ah! And doesn’t it feel good! Well, it certainly feels good for our Heroes of Making Magic, who are casually walking on a one-directional road, fighting imps. Imps are weak and feeble creatures and they are not good at much. However, Heroes enjoy fighting them. For fun, if nothing else.

Our Hero, Ignatius, simply adores imps. He is observing a line of imps, represented as a zero-indexed array of integers *a* of length *n*, where *ai* denotes the number of imps at the *i*-th position. Sometimes, imps can appear out of nowhere. When heroes fight imps, they select a segment of the line, start at one end of the segment, and finish on the other end, without ever exiting the segment. They can move exactly one cell left or right from their current position and when they do so, they defeat one imp on the cell that they moved to, so, the number of imps on that cell decreases by one. This also applies when heroes appear at one end of the segment, at the beginning of their walk.

Their goal is to defeat all imps on the segment, without ever moving to an empty cell in it (without imps), since they would get bored. Since Ignatius loves imps, he doesn’t really want to fight them, so no imps are harmed during the events of this task. However, he would like you to tell him whether it would be possible for him to clear a certain segment of imps in the above mentioned way if he wanted to.

You are given *q* queries, which have two types:

* 1 *a* *b* *k* — denotes that *k* imps appear at each cell from the interval [*a*, *b*]
* 2 *a* *b* - asks whether Ignatius could defeat all imps on the interval [*a*, *b*] in the way described above

**Input**

The first line contains a single integer *n* (1 ≤ *n* ≤ 200 000), the length of the array *a*. The following line contains *n* integers *a*1, *a*2, ..., *an*(0 ≤ *ai* ≤ 5 000), the initial number of imps in each cell. The third line contains a single integer *q* (1 ≤ *q* ≤ 300 000), the number of queries. The remaining *q* lines contain one query each. Each query is provided by integers *a*, *b* and, possibly, *k* (0 ≤ *a* ≤ *b* < *n*, 0 ≤ *k* ≤ 5 000).

**Output**

For each second type of query output 1 if it is possible to clear the segment, and 0 if it is not.

**Example**

**input**

3  
2 2 2  
3  
2 0 2  
1 1 1 1  
2 0 2

**output**

0  
1

**Note**

For the first query, one can easily check that it is indeed impossible to get from the first to the last cell while clearing everything. After we add 1 to the second position, we can clear the segment, for example by moving in the following way: http://codeforces.com/predownloaded/bb/0d/bb0d6152150dfec2cd7e3dc23264a33a6cdd6a20.png.

E. Paint it really, really dark gray

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

I see a pink boar and I want it painted black. Black boars look much more awesome and mighty than the pink ones. Since Jaggy became the ruler of the forest, he has been trying his best to improve the diplomatic relations between the forest region and the nearby ones.

Some other rulers, however, have requested too much in return for peace between their two regions, so he realized he has to resort to intimidation. Once a delegate for diplomatic relations of a neighboring region visits Jaggy’s forest, if they see a whole bunch of black boars, they might suddenly change their mind about attacking Jaggy. Black boars are really scary, after all.

Jaggy’s forest can be represented as a tree (connected graph without cycles) with *n* vertices. Each vertex represents a boar and is colored either black or pink. Jaggy has sent a squirrel to travel through the forest and paint all the boars black. The squirrel, however, is quite unusually trained and while it traverses the graph, it changes the color of every vertex it visits, regardless of its initial color: pink vertices become black and black vertices become pink.

Since Jaggy is too busy to plan the squirrel’s route, he needs your help. He wants you to construct a walk through the tree starting from vertex 1 such that in the end all vertices are black. A walk is a sequence of vertices, such that every consecutive pair has an edge between them in a tree.

**Input**

The first line of input contains integer *n* (2 ≤ *n* ≤ 200 000), denoting the number of vertices in the tree. The following *n* lines contains *n*integers, which represent the color of the nodes.

If the *i*-th integer is 1, if the *i*-th vertex is black and  - 1 if the *i*-th vertex is pink.

Each of the next *n* - 1 lines contains two integers, which represent the indexes of the vertices which are connected by the edge. Vertices are numbered starting with 1.

**Output**

Output path of a squirrel: output a sequence of visited nodes' indexes in order of visiting. In case of all the nodes are initially black, you should print 1. Solution is guaranteed to exist. If there are multiple solutions to the problem you can output any of them provided length of sequence is not longer than 107.

**Example**

**input**

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

**output**

1 4 2 5 2 4 3 4 1 4 1

**Note**

At the beginning squirrel is at node 1 and its color is black. Next steps are as follows:

* From node 1 we walk to node 4 and change its color to pink.
* From node 4 we walk to node 2 and change its color to pink.
* From node 2 we walk to node 5 and change its color to black.
* From node 5 we return to node 2 and change its color to black.
* From node 2 we walk to node 4 and change its color to black.
* We visit node 3 and change its color to black.
* We visit node 4 and change its color to pink.
* We visit node 1 and change its color to pink.
* We visit node 4 and change its color to black.
* We visit node 1 and change its color to black.

D. Dexterina’s Lab

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Dexterina and Womandark have been arch-rivals since they’ve known each other. Since both are super-intelligent teenage girls, they’ve always been trying to solve their disputes in a peaceful and nonviolent way. After god knows how many different challenges they’ve given to one another, their score is equal and they’re both desperately trying to best the other in various games of wits. This time, Dexterina challenged Womandark to a game of Nim.

Nim is a two-player game in which players take turns removing objects from distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects from a single heap. The player who can't make a turn loses. By their agreement, the sizes of piles are selected randomly from the range [0, *x*]. Each pile's size is taken independently from the same probability distribution that is known before the start of the game.

Womandark is coming up with a brand new and evil idea on how to thwart Dexterina’s plans, so she hasn’t got much spare time. She, however, offered you some tips on looking fabulous in exchange for helping her win in Nim. Your task is to tell her what is the probability that the first player to play wins, given the rules as above.

**Input**

The first line of the input contains two integers *n* (1 ≤ *n* ≤ 109) and *x* (1 ≤ *x* ≤ 100) — the number of heaps and the maximum number of objects in a heap, respectively. The second line contains *x* + 1 real numbers, given with up to 6 decimal places each: *P*(0), *P*(1), ... , *P*(*X*). Here, *P*(*i*) is the probability of a heap having exactly *i* objects in start of a game. It's guaranteed that the sum of all *P*(*i*) is equal to 1.

**Output**

Output a single real number, the probability that the first player wins. The answer will be judged as correct if it differs from the correct answer by at most 10- 6.

**Example**

**input**

2 2  
0.500000 0.250000 0.250000

**output**

0.62500000

C. Potions Homework

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Harry Water, Ronaldo, Her-my-oh-knee and their friends have started a new school year at their MDCS School of Speechcraft and Misery. At the time, they are very happy to have seen each other after a long time. The sun is shining, birds are singing, flowers are blooming, and their Potions class teacher, professor Snipe is sulky as usual. Due to his angst fueled by disappointment in his own life, he has given them a lot of homework in Potions class.

Each of the *n* students has been assigned a single task. Some students do certain tasks faster than others. Thus, they want to redistribute the tasks so that each student still does exactly one task, and that all tasks are finished. Each student has their own laziness level, and each task has its own difficulty level. Professor Snipe is trying hard to improve their work ethics, so each student’s laziness level is equal to their task’s difficulty level. Both sets of values are given by the sequence *a*, where *ai* represents both the laziness level of the *i*-th student and the difficulty of his task.

The time a student needs to finish a task is equal to the product of their laziness level and the task’s difficulty. They are wondering, what is the minimum possible total time they must spend to finish all tasks if they distribute them in the optimal way. Each person should receive one task and each task should be given to one person. Print the answer modulo 10 007.

**Input**

The first line of input contains integer *n* (1 ≤ *n* ≤ 100 000) — the number of tasks. The next *n* lines contain exactly one integer number *ai* (1 ≤ *ai* ≤ 100 000) — both the difficulty of the initial task and the laziness of the *i*-th students.

**Output**

Print the minimum total time to finish all tasks modulo 10 007.

**Example**

**input**

2  
1  
3

**output**

6

**Note**

In the first sample, if the students switch their tasks, they will be able to finish them in 3 + 3 = 6 time units.

B. R3D3’s Summer Adventure

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

R3D3 spent some time on an internship in MDCS. After earning enough money, he decided to go on a holiday somewhere far, far away. He enjoyed suntanning, drinking alcohol-free cocktails and going to concerts of popular local bands. While listening to "The White Buttons" and their hit song "Dacan the Baker", he met another robot for whom he was sure is the love of his life. Well, his summer, at least. Anyway, R3D3 was too shy to approach his potential soulmate, so he decided to write her a love letter. However, he stumbled upon a problem. Due to a terrorist threat, the Intergalactic Space Police was monitoring all letters sent in the area. Thus, R3D3 decided to invent his own alphabet, for which he was sure his love would be able to decipher.

There are *n* letters in R3D3’s alphabet, and he wants to represent each letter as a sequence of '0' and '1', so that no letter’s sequence is a prefix of another letter's sequence. Since the Intergalactic Space Communications Service has lately introduced a tax for invented alphabets, R3D3 must pay a certain amount of money for each bit in his alphabet’s code (check the sample test for clarifications). He is too lovestruck to think clearly, so he asked you for help.

Given the costs *c*0 and *c*1 for each '0' and '1' in R3D3’s alphabet, respectively, you should come up with a coding for the alphabet (with properties as above) with minimum total cost.

**Input**

The first line of input contains three integers *n* (2 ≤ *n* ≤ 108), *c*0 and *c*1 (0 ≤ *c*0, *c*1 ≤ 108) — the number of letters in the alphabet, and costs of '0' and '1', respectively.

**Output**

Output a single integer — minimum possible total a cost of the whole alphabet.

**Example**

**input**

4 1 2

**output**

12

**Note**

There are 4 letters in the alphabet. The optimal encoding is "00", "01", "10", "11". There are 4 zeroes and 4 ones used, so the total cost is 4·1 + 4·2 = 12.

A. Festival Organization

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

The Prodiggers are quite a cool band and for this reason, they have been the surprise guest at the ENTER festival for the past 80 years. At the beginning of their careers, they weren’t so successful, so they had to spend time digging channels to earn money; hence the name. Anyway, they like to tour a lot and have surprising amounts of energy to do extremely long tours. However, they hate spending two consecutive days without having a concert, so they would like to avoid it.

A *tour* is defined by a sequence of concerts and days-off. You need to count in how many ways The Prodiggers can select *k* different tours of the same length between *l* and *r*.

For example if *k* = 2, *l* = 1 and *r* = 2, if we define concert day as {1} and day-off as {0}, here are all possible tours: {0}, {1}, {00}, {01}, {10}, {11}. But tour 00 can not be selected because it has 2 days-off in a row. Now, we need to count in how many ways we can select *k* = 2 tours **of the same length** in range [1;2]. Here they are: {0,1}; {01,10}; {01,11}; {10,11}.

Since their schedule is quite busy, they want you to tell them in how many ways can do that, modulo 1 000 000 007 (109 + 7).

**Input**

The first line of the input contains three integers *k*, *l* and *r* (1 ≤ *k* ≤ 200, 1 ≤ *l* ≤ *r* ≤ 1018).

**Output**

Output a single number: the number of ways to select *k* different tours of the same length, modulo 1 000 000 007.

**Example**

**input**

1 1 2

**output**

5

B. Complete the Word

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder loves to read the dictionary. He thinks that a word is *nice* if there exists a **substring** (contiguous segment of letters) of it of length 26 where each letter of English alphabet appears exactly once. In particular, if the string has length strictly less than 26, no such substring exists and thus it is not nice.

Now, ZS the Coder tells you a word, where some of its letters are missing as he forgot them. He wants to determine if it is possible to fill in the missing letters so that the resulting word is nice. If it is possible, he needs you to find an example of such a word as well. Can you help him?

**Input**

The first and only line of the input contains a single string *s* (1 ≤ |*s*| ≤ 50 000), the word that ZS the Coder remembers. Each character of the string is the uppercase letter of English alphabet ('A'-'Z') or is a question mark ('?'), where the question marks denotes the letters that ZS the Coder can't remember.

**Output**

If there is no way to replace all the question marks with **uppercase letters** such that the resulting word is nice, then print  - 1 in the only line.

Otherwise, print a string which denotes a possible nice word that ZS the Coder learned. This string should match the string from the input, except for the question marks replaced with uppercase English letters.

If there are multiple solutions, you may print any of them.

**Examples**

**input**

ABC??FGHIJK???OPQR?TUVWXY?

**output**

ABCDEFGHIJKLMNOPQRZTUVWXYS

**input**

WELCOMETOCODEFORCESROUNDTHREEHUNDREDANDSEVENTYTWO

**output**

-1

**input**

??????????????????????????

**output**

MNBVCXZLKJHGFDSAQPWOEIRUYT

**input**

AABCDEFGHIJKLMNOPQRSTUVW??M

**output**

-1

**Note**

In the first sample case, ABCDEFGHIJKLMNOPQRZTUVWXYS is a valid answer beacuse it contains a substring of length 26 (the whole string in this case) which contains all the letters of the English alphabet exactly once. Note that there are many possible solutions, such as ABCDEFGHIJKLMNOPQRSTUVWXYZ or ABCEDFGHIJKLMNOPQRZTUVWXYS.

In the second sample case, there are no missing letters. In addition, the given string does not have a substring of length 26 that contains all the letters of the alphabet, so the answer is  - 1.

In the third sample case, any string of length 26 that contains all letters of the English alphabet fits as an answer.

A. Crazy Computer

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder is coding on a crazy computer. If you don't type in a word for a *c* consecutive seconds, everything you typed disappear!

More formally, if you typed a word at second *a* and then the next word at second *b*, then if *b* - *a* ≤ *c*, just the new word is appended to other words on the screen. If *b* - *a* > *c*, then everything on the screen disappears and after that the word you have typed appears on the screen.

For example, if *c* = 5 and you typed words at seconds 1, 3, 8, 14, 19, 20 then at the second 8 there will be 3 words on the screen. After that, everything disappears at the second 13 because nothing was typed. At the seconds 14 and 19 another two words are typed, and finally, at the second 20, one more word is typed, and a total of 3 words remain on the screen.

You're given the times when ZS the Coder typed the words. Determine how many words remain on the screen after he finished typing everything.

**Input**

The first line contains two integers *n* and *c* (1 ≤ *n* ≤ 100 000, 1 ≤ *c* ≤ 109) — the number of words ZS the Coder typed and the crazy computer delay respectively.

The next line contains *n* integers *t*1, *t*2, ..., *tn* (1 ≤ *t*1 < *t*2 < ... < *tn* ≤ 109), where *ti* denotes the second when ZS the Coder typed the *i*-th word.

**Output**

Print a single positive integer, the number of words that remain on the screen after all *n* words was typed, in other words, at the second *tn*.

**Examples**

**input**

6 5  
1 3 8 14 19 20

**output**

3

**input**

6 1  
1 3 5 7 9 10

**output**

2

**Note**

The first sample is already explained in the problem statement.

For the second sample, after typing the first word at the second 1, it disappears because the next word is typed at the second 3 and 3 - 1 > 1. Similarly, only 1 word will remain at the second 9. Then, a word is typed at the second 10, so there will be two words on the screen, as the old word won't disappear because 10 - 9 ≤ 1.

E. Complete the Permutations

715E

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder is given two permutations *p* and *q* of {1, 2, ..., *n*}, but some of their elements are replaced with 0. The *distance* between two permutations *p* and *q* is defined as the minimum number of moves required to turn *p* into *q*. A move consists of swapping exactly 2elements of *p*.

ZS the Coder wants to determine the number of ways to replace the zeros with positive integers from the set {1, 2, ..., *n*} such that *p*and *q* are permutations of {1, 2, ..., *n*} and the distance between *p* and *q* is exactly *k*.

ZS the Coder wants to find the answer for all 0 ≤ *k* ≤ *n* - 1. Can you help him?

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 250) — the number of elements in the permutations.

The second line contains *n* integers, *p*1, *p*2, ..., *pn* (0 ≤ *pi* ≤ *n*) — the permutation *p*. It is guaranteed that there is at least one way to replace zeros such that *p* is a permutation of {1, 2, ..., *n*}.

The third line contains *n* integers, *q*1, *q*2, ..., *qn* (0 ≤ *qi* ≤ *n*) — the permutation *q*. It is guaranteed that there is at least one way to replace zeros such that *q* is a permutation of {1, 2, ..., *n*}.

**Output**

Print *n* integers, *i*-th of them should denote the answer for *k* = *i* - 1. Since the answer may be quite large, and ZS the Coder loves weird primes, print them modulo 998244353 = 223·7·17 + 1, which is a prime.

**Examples**

**input**

3  
1 0 0  
0 2 0

**output**

1 2 1

**input**

4  
1 0 0 3  
0 0 0 4

**output**

0 2 6 4

**input**

6  
1 3 2 5 4 6  
6 4 5 1 0 0

**output**

0 0 0 0 1 1

**input**

4  
1 2 3 4  
2 3 4 1

**output**

0 0 0 1

**Note**

In the first sample case, there is the only way to replace zeros so that it takes 0 swaps to convert *p* into *q*, namely *p* = (1, 2, 3), *q* = (1, 2, 3).

There are two ways to replace zeros so that it takes 1 swap to turn *p* into *q*. One of these ways is *p* = (1, 2, 3), *q* = (3, 2, 1), then swapping 1 and 3 from *p* transform it into *q*. The other way is *p* = (1, 3, 2), *q* = (1, 2, 3). Swapping 2 and 3 works in this case.

Finally, there is one way to replace zeros so that it takes 2 swaps to turn *p* into *q*, namely *p* = (1, 3, 2), *q* = (3, 2, 1). Then, we can transform *p* into *q* like following: http://codeforces.com/predownloaded/bd/3a/bd3a9fdb5d1df7466b5b4cedbadd4149ac8cb0c4.png.

D. Create a Maze

time limit per test

2 seconds

memory limit per test

256 megabytes

input

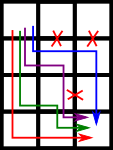
standard input

output

standard output

ZS the Coder loves mazes. Your job is to create one so that he can play with it. A maze consists of *n* × *m* rooms, and the rooms are arranged in *n* rows (numbered from the top to the bottom starting from 1) and *m* columns (numbered from the left to the right starting from 1). The room in the *i*-th row and *j*-th column is denoted by (*i*, *j*). A player starts in the room (1, 1) and wants to reach the room (*n*, *m*).

Each room has four doors (except for ones at the maze border), one on each of its walls, and two adjacent by the wall rooms shares the same door. Some of the doors are locked, which means it is impossible to pass through the door. For example, if the door connecting (*i*, *j*) and (*i*, *j* + 1) is locked, then we can't go from (*i*, *j*) to (*i*, *j* + 1). Also, one can only travel between the rooms downwards (from the room (*i*, *j*) to the room (*i* + 1, *j*)) or rightwards (from the room (*i*, *j*) to the room (*i*, *j* + 1)) provided the corresponding door is not locked.

This image represents a maze with some doors locked. The colored arrows denotes all the possible paths while a red cross denotes a locked door.

ZS the Coder considers a maze to have *difficulty* *x* if there is exactly *x* ways of travelling from the room (1, 1) to the room (*n*, *m*). Two ways are considered different if they differ by the sequence of rooms visited while travelling.

Your task is to create a maze such that its difficulty is exactly equal to *T*. In addition, ZS the Coder doesn't like large mazes, so the size of the maze and the number of locked doors are limited. Sounds simple enough, right?

**Input**

The first and only line of the input contains a single integer *T* (1 ≤ *T* ≤ 1018), the difficulty of the required maze.

**Output**

The first line should contain two integers *n* and *m* (1 ≤ *n*, *m* ≤ 50) — the number of rows and columns of the maze respectively.

The next line should contain a single integer *k* (0 ≤ *k* ≤ 300) — the number of locked doors in the maze.

Then, *k* lines describing locked doors should follow. Each of them should contain four integers, *x*1, *y*1, *x*2, *y*2. This means that the door connecting room (*x*1, *y*1) and room (*x*2, *y*2) is locked. Note that room (*x*2, *y*2) should be adjacent either to the right or to the bottom of (*x*1, *y*1), i.e. *x*2 + *y*2 should be equal to *x*1 + *y*1 + 1. There should not be a locked door that appears twice in the list.

It is guaranteed that at least one solution exists. If there are multiple solutions, print any of them.

**Examples**

**input**

3

**output**

3 2  
0

**input**

4

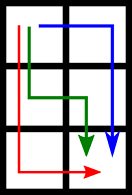
**output**

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

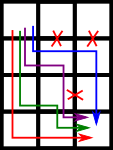
**Note**

Here are how the sample input and output looks like. The colored arrows denotes all the possible paths while a red cross denotes a locked door.

In the first sample case:



In the second sample case:



C. Digit Tree

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder has a large tree. It can be represented as an undirected connected graph of *n* vertices numbered from 0 to *n* - 1 and *n* - 1edges between them. There is a single **nonzero** digit written on each edge.

One day, ZS the Coder was bored and decided to investigate some properties of the tree. He chose a positive integer *M*, which is **coprime** to 10, i.e. http://codeforces.com/predownloaded/cc/01/cc010bb0783dfd849110ad2d02c83e233a8392af.png.

ZS consider an **ordered pair** of distinct vertices (*u*, *v*) *interesting* when if he would follow the shortest path from vertex *u* to vertex *v* and write down all the digits he encounters on his path in the same order, he will get a decimal representaion of an integer divisible by *M*.

Formally, ZS consider an ordered pair of distinct vertices (*u*, *v*) interesting if the following states true:

* Let *a*1 = *u*, *a*2, ..., *ak* = *v* be the sequence of vertices on the shortest path from *u* to *v* in the order of encountering them;
* Let *di* (1 ≤ *i* < *k*) be the digit written on the edge between vertices *ai* and *ai*+ 1;
* The integer http://codeforces.com/predownloaded/3b/00/3b0042880cc54c29eb7a83206104594076ddcebd.png is divisible by *M*.

Help ZS the Coder find the number of interesting pairs!

**Input**

The first line of the input contains two integers, *n* and *M* (2 ≤ *n* ≤ 100 000, 1 ≤ *M* ≤ 109, http://codeforces.com/predownloaded/28/53/28537113bb89960624c38c889f25be35f15cb4b4.png) — the number of vertices and the number ZS has chosen respectively.

The next *n* - 1 lines contain three integers each. *i*-th of them contains *ui*, *vi* and *wi*, denoting an edge between vertices *ui* and *vi* with digit *wi* written on it (0 ≤ *ui*, *vi* < *n*,  1 ≤ *wi* ≤ 9).

**Output**

Print a single integer — the number of interesting (by ZS the Coder's consideration) pairs.

**Examples**

**input**

6 7  
0 1 2  
4 2 4  
2 0 1  
3 0 9  
2 5 7

**output**

7

**input**

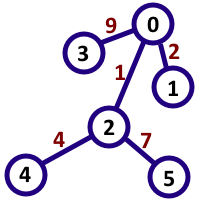
5 11  
1 2 3  
2 0 3  
3 0 3  
4 3 3

**output**

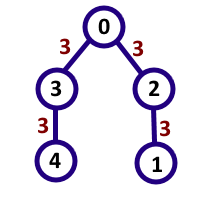
8

**Note**

In the first sample case, the interesting pairs are (0, 4), (1, 2), (1, 5), (3, 2), (2, 5), (5, 2), (3, 5). The numbers that are formed by these pairs are 14, 21, 217, 91, 7, 7, 917 respectively, which are all multiples of 7. Note that (2, 5) and (5, 2) are considered different.



In the second sample case, the interesting pairs are (4, 0), (0, 4), (3, 2), (2, 3), (0, 1), (1, 0), (4, 1), (1, 4), and 6 of these pairs give the number 33 while 2 of them give the number 3333, which are all multiples of 11.



B. Complete The Graph

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder has drawn an undirected graph of *n* vertices numbered from 0 to *n* - 1 and *m* edges between them. Each edge of the graph is weighted, each weight is a **positive integer**.

The next day, ZS the Coder realized that some of the weights were erased! So he wants to reassign **positive integer** weight to each of the edges which weights were erased, so that the length of the shortest path between vertices *s* and *t* in the resulting graph is exactly *L*. Can you help him?

**Input**

The first line contains five integers *n*, *m*, *L*, *s*, *t* (2 ≤ *n* ≤ 1000,  1 ≤ *m* ≤ 10 000,  1 ≤ *L* ≤ 109,  0 ≤ *s*, *t* ≤ *n* - 1,  *s* ≠ *t*) — the number of vertices, number of edges, the desired length of shortest path, starting vertex and ending vertex respectively.

Then, *m* lines describing the edges of the graph follow. *i*-th of them contains three integers, *ui*, *vi*, *wi*(0 ≤ *ui*, *vi* ≤ *n* - 1,  *ui* ≠ *vi*,  0 ≤ *wi* ≤ 109). *ui* and *vi* denote the endpoints of the edge and *wi* denotes its weight. If *wi* is equal to 0 then the weight of the corresponding edge was erased.

It is guaranteed that there is at most one edge between any pair of vertices.

**Output**

Print "NO" (without quotes) in the only line if it's not possible to assign the weights in a required way.

Otherwise, print "YES" in the first line. Next *m* lines should contain the edges of the resulting graph, with weights assigned to edges which weights were erased. *i*-th of them should contain three integers *ui*, *vi* and *wi*, denoting an edge between vertices *ui* and *vi* of weight *wi*. The edges of the new graph must coincide with the ones in the graph from the input. The weights that were not erased must remain unchanged whereas the new weights can be any **positive integer** not exceeding 1018.

The order of the edges in the output doesn't matter. The length of the shortest path between *s* and *t* must be equal to *L*.

If there are multiple solutions, print any of them.

**Examples**

**input**

5 5 13 0 4  
0 1 5  
2 1 2  
3 2 3  
1 4 0  
4 3 4

**output**

YES  
0 1 5  
2 1 2  
3 2 3  
1 4 8  
4 3 4

**input**

2 1 123456789 0 1  
0 1 0

**output**

YES  
0 1 123456789

**input**

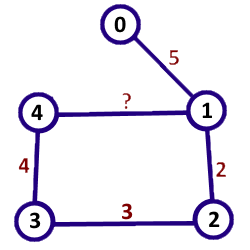
2 1 999999999 1 0  
0 1 1000000000

**output**

NO

**Note**

Here's how the graph in the first sample case looks like :



In the first sample case, there is only one missing edge weight. Placing the weight of 8 gives a shortest path from 0 to 4 of length 13.

In the second sample case, there is only a single edge. Clearly, the only way is to replace the missing weight with 123456789.

In the last sample case, there is no weights to assign but the length of the shortest path doesn't match the required value, so the answer is "NO".

A. Plus and Square Root

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder is playing a game. There is a number displayed on the screen and there are two buttons, ' + ' (plus) and 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' (square root). Initially, the number 2 is displayed on the screen. There are *n* + 1 levels in the game and ZS the Coder start at the level 1.

When ZS the Coder is at level *k*, he can :

1. *Press the ' + ' button*. This increases the number on the screen by exactly *k*. So, if the number on the screen was *x*, it becomes *x* + *k*.
2. *Press the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button*. Let the number on the screen be *x*. After pressing this button, the number becomes http://codeforces.com/predownloaded/f6/88/f6882b74756d5243f4ab3ee90632620b752befc1.png. After that, ZS the Coder levels up, so his current level becomes *k* + 1. This button can only be pressed when *x* is a **perfect square**, i.e. *x* = *m*2 for some positive integer *m*.

Additionally, after each move, if ZS the Coder is at level *k*, and the number on the screen is *m*, then ***m* must be a multiple of *k***. Note that this condition is only checked after performing the press. For example, if ZS the Coder is at level 4 and current number is 100, he presses the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button and the number turns into 10. Note that at this moment, 10 is not divisible by 4, but this press is still valid, because after it, ZS the Coder is at level 5, and 10 is divisible by 5.

ZS the Coder needs your help in beating the game — he wants to reach level *n* + 1. In other words, he needs to press the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button *n*times. Help him determine the number of times he should press the ' + ' button before pressing the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button at each level.

Please note that ZS the Coder wants to find just any sequence of presses allowing him to reach level *n* + 1, but not necessarily a sequence minimizing the number of presses.

**Input**

The first and only line of the input contains a single integer *n* (1 ≤ *n* ≤ 100 000), denoting that ZS the Coder wants to reach level *n* + 1.

**Output**

Print *n* non-negative integers, one per line. *i*-th of them should be equal to the number of times that ZS the Coder needs to press the ' + ' button before pressing the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button at level *i*.

Each number in the output should not exceed 1018. However, the number on the screen **can be greater** than 1018.

It is guaranteed that at least one solution exists. If there are multiple solutions, print any of them.

**Examples**

**input**

3

**output**

14  
16  
46

**input**

2

**output**

999999999999999998  
44500000000

**input**

4

**output**

2  
17  
46  
97

**Note**

In the first sample case:

On the first level, ZS the Coder pressed the ' + ' button 14 times (and the number on screen is initially 2), so the number became 2 + 14·1 = 16. Then, ZS the Coder pressed the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button, and the number became http://codeforces.com/predownloaded/c8/8e/c88ee2affeb9ab59e5515446424e7ccbf1477604.png.

After that, on the second level, ZS pressed the ' + ' button 16 times, so the number becomes 4 + 16·2 = 36. Then, ZS pressed the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button, levelling up and changing the number into http://codeforces.com/predownloaded/e5/fb/e5fbffcf9d4b46603ed3dda2a4397f72077c1560.png.

After that, on the third level, ZS pressed the ' + ' button 46 times, so the number becomes 6 + 46·3 = 144. Then, ZS pressed the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button, levelling up and changing the number into http://codeforces.com/predownloaded/63/f5/63f5b7284f624ac179af9c3568b1e60b3d1f0ed3.png.

Note that 12 is indeed divisible by 4, so ZS the Coder can reach level 4.

Also, note that pressing the ' + ' button 10 times on the third level before levelling up does not work, because the number becomes 6 + 10·3 = 36, and when the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button is pressed, the number becomes http://codeforces.com/predownloaded/e5/fb/e5fbffcf9d4b46603ed3dda2a4397f72077c1560.png and ZS the Coder is at Level 4. However, 6 is not divisible by 4 now, so this is **not a valid solution.**

In the second sample case:

On the first level, ZS the Coder pressed the ' + ' button 999999999999999998 times (and the number on screen is initially 2), so the number became 2 + 999999999999999998·1 = 1018. Then, ZS the Coder pressed the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button, and the number became http://codeforces.com/predownloaded/15/9d/159d231369998a4595804f8c92080d0fe04a69b2.png.

After that, on the second level, ZS pressed the ' + ' button 44500000000 times, so the number becomes 109 + 44500000000·2 = 9·1010. Then, ZS pressed the 'http://codeforces.com/predownloaded/8e/78/8e789c13c3006eb096030764e2b4ab064c26c2ed.png' button, levelling up and changing the number into http://codeforces.com/predownloaded/0b/dd/0bddda0267e6ff6b3eb3366c1a2c030ef75e42ab.png.

Note that 300000 is a multiple of 3, so ZS the Coder can reach level 3.

B. Filya and Homework

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Today, hedgehog Filya went to school for the very first time! Teacher gave him a homework which Filya was unable to complete without your help.

Filya is given an array of non-negative integers *a*1, *a*2, ..., *an*. First, he pick an integer *x* and then he adds *x* to some elements of the array (no more than once), subtract *x* from some other elements (also, no more than once) and do no change other elements. He wants all elements of the array to be equal.

Now he wonders if it's possible to pick such integer *x* and change some elements of the array using this *x* in order to make all elements equal.

**Input**

The first line of the input contains an integer *n* (1 ≤ *n* ≤ 100 000) — the number of integers in the Filya's array. The second line contains *n* integers *a*1, *a*2, ..., *an* (0 ≤ *ai* ≤ 109) — elements of the array.

**Output**

If it's impossible to make all elements of the array equal using the process given in the problem statement, then print "NO" (without quotes) in the only line of the output. Otherwise print "YES" (without quotes).

**Examples**

**input**

5  
1 3 3 2 1

**output**

YES

**input**

5  
1 2 3 4 5

**output**

NO

**Note**

In the first sample Filya should select *x* = 1, then add it to the first and the last elements of the array and subtract from the second and the third elements.

A. Meeting of Old Friends

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Today an outstanding event is going to happen in the forest — hedgehog Filya will come to his old fried Sonya!

Sonya is an owl and she sleeps during the day and stay awake from minute *l*1 to minute *r*1 inclusive. Also, during the minute *k* she prinks and is unavailable for Filya.

Filya works a lot and he plans to visit Sonya from minute *l*2 to minute *r*2 inclusive.

Calculate the number of minutes they will be able to spend together.

**Input**

The only line of the input contains integers *l*1, *r*1, *l*2, *r*2 and *k* (1 ≤ *l*1, *r*1, *l*2, *r*2, *k* ≤ 1018, *l*1 ≤ *r*1, *l*2 ≤ *r*2), providing the segments of time for Sonya and Filya and the moment of time when Sonya prinks.

**Output**

Print one integer — the number of minutes Sonya and Filya will be able to spend together.

**Examples**

**input**

1 10 9 20 1

**output**

2

**input**

1 100 50 200 75

**output**

50

**Note**

In the first sample, they will be together during minutes 9 and 10.

In the second sample, they will be together from minute 50 to minute 74 and from minute 76 to minute 100.

E. Sonya Partymaker

time limit per test

1.5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Owl Sonya decided to become a partymaker. To train for this role she gather all her owl friends in the country house. There are *m* chairs located in a circle and consequently numbered with integers from 1 to *m*. Thus, chairs *i* and *i* + 1 are neighbouring for all *i* from 1 to *m* - 1. Chairs 1 and *m* are also neighbouring. Some chairs are occupied by her friends. There are *n* friends in total. No two friends occupy the same chair. Rules are the following:

1. Each participant removes from the game the chair he is currently sitting on.
2. Each of the participants choose a direction that she will follow: clockwise (indices increase, from *m* goes to 1) and counter-clockwise (indices decrease, from 1 goes to *m*). This direction may coincide or be different for any pair of owls.
3. Each turn all guests move one step in the chosen directions. If some guest move to the position with a chair there, he removes this chair from the game.
4. Game ends if there are no more chairs left in the game.

Owls are very busy and want to get rid of the game as soon as possible. They cooperate to pick the direction. Your goal is to find the minimum number o moves required to finish the game.

**Input**

The first line of the input contains a single integer *m* (1 ≤ *m* ≤ 109) — the length of the circle.

The second line contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the number of friends.

Last line contains an increasing sequence of *n* integers *ai* (1 ≤ *ai* ≤ *m*) — initial positions of all owls.

**Output**

Print the minimum number of move required to finish the game. Note, that 0 also may be an answer.

**Examples**

**input**

6  
3  
1 3 5

**output**

1

**input**

6  
2  
1 6

**output**

2

**input**

406  
6  
1 2 3 204 205 206

**output**

100

**Note**

In the first sample, it's possible if all owls will move clockwise, i.e. in the direction of increasing indices.

In the sample, first owl has to move clockwise, while the second — counterclockwise.

In the third sample, the first and the fourth owls should move counterclockwise, while the third and the sixth — clockwise. The second and the firth may move in any direction.

D. Animals and Puzzle

time limit per test

5 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Owl Sonya gave a huge lake puzzle of size *n* × *m* to hedgehog Filya as a birthday present. Friends immediately started to assemble the puzzle, but some parts of it turned out to be empty — there was no picture on them. Parts with picture on it are denoted by 1, while empty parts are denoted by 0. Rows of the puzzle are numbered from top to bottom with integers from 1 to *n*, while columns are numbered from left to right with integers from 1 to *m*.

Animals decided to complete the picture and play with it, as it might be even more fun! Owl and hedgehog ask each other some queries. Each query is provided by four integers *x*1, *y*1, *x*2, *y*2 which define the rectangle, where (*x*1, *y*1) stands for the coordinates of the up left cell of the rectangle, while (*x*2, *y*2) stands for the coordinates of the bottom right cell. The answer to the query is the size of the maximum **square** consisting of picture parts only (only parts denoted by 1) and located fully inside the query rectangle.

Help Sonya and Filya answer *t* queries.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 1000) — sizes of the puzzle.

Each of the following *n* lines contains *m* integers *aij*. Each of them is equal to 1 if the corresponding cell contains a picture and 0 if it's empty.

Next line contains an integer *t* (1 ≤ *t* ≤ 1 000 000) — the number of queries.

Then follow *t* lines with queries' descriptions. Each of them contains four integers *x*1, *y*1, *x*2, *y*2 (1 ≤ *x*1 ≤ *x*2 ≤ *n*, 1 ≤ *y*1 ≤ *y*2 ≤ *m*) — coordinates of the up left and bottom right cells of the query rectangle.

**Output**

Print *t* lines. The *i*-th of them should contain the maximum size of the square consisting of 1-s and lying fully inside the query rectangle.

**Example**

**input**

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

**output**

1  
1  
1  
2  
2

C. Sonya and Problem Wihtout a Legend

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Sonya was unable to think of a story for this problem, so here comes the formal description.

You are given the array containing *n* positive integers. At one turn you can pick any element and increase or decrease it by 1. The goal is the make the array strictly increasing by making the minimum possible number of operations. You are allowed to change elements in any way, they can become negative or equal to 0.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 3000) — the length of the array.

Next line contains *n* integer *ai* (1 ≤ *ai* ≤ 109).

**Output**

Print the minimum number of operation required to make the array strictly increasing.

**Examples**

**input**

7  
2 1 5 11 5 9 11

**output**

9

**input**

5  
5 4 3 2 1

**output**

12

**Note**

In the first sample, the array is going to look as follows:

2 3 5 6 7 9 11

|2 - 2| + |1 - 3| + |5 - 5| + |11 - 6| + |5 - 7| + |9 - 9| + |11 - 11| = 9

And for the second sample:

1 2 3 4 5

|5 - 1| + |4 - 2| + |3 - 3| + |2 - 4| + |1 - 5| = 12

B. Searching Rectangles

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Filya just learned new geometry object — rectangle. He is given a field consisting of *n* × *n* unit cells. Rows are numbered from bottom to top with integer from 1 to *n*. Columns are numbered from left to right with integers from 1 to *n*. Cell, located at the intersection of the row *r* and column *c* is denoted as (*r*, *c*). Filya has painted two rectangles, such that their sides are parallel to coordinate axes and each cell lies fully inside or fully outside each of them. Moreover, no cell lies in both rectangles.

Later, hedgehog Filya became interested in the location of his rectangles but was unable to find the sheet of paper they were painted on. They were taken by Sonya and now she wants to play a little game with Filya. He tells her a query rectangle and she replies with the number of initial rectangles that lie **fully inside** the given query rectangle. The query rectangle should match the same conditions as initial rectangles. Rectangle lies fully inside the query if each o its cells lies inside the query.

Filya knows Sonya really well, so is sure that if he asks more than 200 questions she will stop to reply.

**Input**

The first line of the input contains an integer *n* (2 ≤ *n* ≤ 216) — size of the field.

For each query an integer between 0 and 2 is returned — the number of initial rectangles that lie fully inside the query rectangle.

**Output**

To make a query you have to print "? *x*1 *y*1 *x*2 *y*2" (without quotes) (1 ≤ *x*1 ≤ *x*2 ≤ *n*, 1 ≤ *y*1 ≤ *y*2 ≤ *n*), where (*x*1, *y*1) stands for the position of the bottom left cell of the query and (*x*2, *y*2) stands for the up right cell of the query. You are allowed to ask no more than 200queries. After each query you should perform "flush" operation and read the answer.

In case you suppose you've already determined the location of two rectangles (or run out of queries) you should print "! *x*11 *y*11 *x*12 *y*12*x*21 *y*21 *x*22 *y*22" (without quotes), where first four integers describe the bottom left and up right cells of the first rectangle, and following four describe the corresponding cells of the second rectangle. You can print the rectangles in an arbitrary order. After you have printed the answer, print the end of the line and perform "flush". Your program should terminate immediately after it print the answer.

**Interaction**

To flush you can use (just after printing an integer and end-of-line):

* fflush(stdout) in C++;
* System.out.flush() in Java;
* stdout.flush() in Python;
* flush(output) in Pascal;
* See the documentation for other languages.

You will get the Wrong Answer verdict if you ask more than 200 queries, or if you print an incorrect coordinates.

You will get the Idleness Limit Exceeded verdict if you don't print anything (but you should) or if you forget about flushing the output (more info below).

**Hacking.**

The first line should contain an integer *n* (2 ≤ *n* ≤ 216).

The second line should contain four integers *x*1, *y*1, *x*2, *y*2 (1 ≤ *x*1 ≤ *x*2 ≤ *n*, 1 ≤ *y*1 ≤ *y*2 ≤ *n*) — the description of the first rectangle.

The third line contains the description of the second rectangle in the similar way.

**Example**

**input**

5  
2  
1  
0  
1  
1  
1  
0  
1

**output**

? 1 1 5 5  
? 1 1 3 3  
? 1 1 3 1  
? 2 2 2 2  
? 3 3 5 5  
? 3 3 3 5  
? 3 3 3 4  
? 3 4 3 5  
! 2 2 2 2 3 4 3 5

A. Sonya and Queries

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Today Sonya learned about long integers and invited all her friends to share the fun. Sonya has an initially empty multiset with integers. Friends give her *t* queries, each of one of the following type:

1. +  *ai* — add non-negative integer *ai* to the multiset. Note, that she has a multiset, thus there may be many occurrences of the same integer.
2. -  *ai* — delete a single occurrence of non-negative integer *ai* from the multiset. It's guaranteed, that there is at least one *ai* in the multiset.
3. ? *s* — count the number of integers in the multiset (with repetitions) that match some pattern *s* consisting of 0 and 1. In the pattern, 0 stands for the even digits, while 1 stands for the odd. Integer *x* matches the pattern *s*, if the parity of the *i*-th from the right digit in decimal notation matches the *i*-th from the right digit of the pattern. If the pattern is shorter than this integer, it's supplemented with 0-s from the left. Similarly, if the integer is shorter than the pattern its decimal notation is supplemented with the 0-s from the left.

For example, if the pattern is *s* = 010, than integers 92, 2212, 50 and 414 match the pattern, while integers 3, 110, 25 and 1030 do not.

**Input**

The first line of the input contains an integer *t* (1 ≤ *t* ≤ 100 000) — the number of operation Sonya has to perform.

Next *t* lines provide the descriptions of the queries in order they appear in the input file. The *i*-th row starts with a character *ci* — the type of the corresponding operation. If *ci* is equal to '+' or '-' then it's followed by a space and an integer *ai* (0 ≤ *ai* < 1018) given without leading zeroes (unless it's 0). If *ci* equals '?' then it's followed by a space and a sequence of zeroes and onse, giving the pattern of length no more than 18.

It's guaranteed that there will be at least one query of type '?'.

It's guaranteed that any time some integer is removed from the multiset, there will be at least one occurrence of this integer in it.

**Output**

For each query of the third type print the number of integers matching the given pattern. Each integer is counted as many times, as it appears in the multiset at this moment of time.

**Examples**

**input**

12  
+ 1  
+ 241  
? 1  
+ 361  
- 241  
? 0101  
+ 101  
? 101  
- 101  
? 101  
+ 4000  
? 0

**output**

2  
1  
2  
1  
1

**input**

4  
+ 200  
+ 200  
- 200  
? 0

**output**

1

**Note**

Consider the integers matching the patterns from the queries of the third type. Queries are numbered in the order they appear in the input.

1. 1 and 241.
2. 361.
3. 101 and 361.
4. 361.
5. 4000.

E. Memory and Casinos

time limit per test

4 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

There are *n* casinos lined in a row. If Memory plays at casino *i*, he has probability *pi* to win and move to the casino on the right (*i* + 1) or exit the row (if *i* = *n*), and a probability 1 - *pi* to lose and move to the casino on the left (*i* - 1) or also exit the row (if *i* = 1).

We say that Memory *dominates* on the interval *i*... *j* if he completes a walk such that,

* He starts on casino *i*.
* He never looses in casino *i*.
* He finishes his walk by winning in casino *j*.

Note that Memory can still walk left of the 1-st casino and right of the casino *n* and that always finishes the process.

Now Memory has some requests, in one of the following forms:

* 1 *i* *a* *b*: Set http://codeforces.com/predownloaded/d6/78/d6789e958a223b18aa230b4f10a19d2d41df1eae.png.
* 2 *l* *r*: Print the probability that Memory will *dominate* on the interval *l*... *r*, i.e. compute the probability that Memory will first leave the segment *l*... *r* after winning at casino *r*, if she starts in casino *l*.

It is guaranteed that at any moment of time *p* is a **non-decreasing sequence**, i.e. *pi* ≤ *pi*+ 1 for all *i* from 1 to *n* - 1.

Please help Memory by answering all his requests!

**Input**

The first line of the input contains two integers *n* and *q*(1 ≤ *n*, *q* ≤ 100 000),  — number of casinos and number of requests respectively.

The next *n* lines each contain integers *ai* and *bi* (1 ≤ *ai* < *bi* ≤ 109)  — http://codeforces.com/predownloaded/75/56/7556e975c5fa2f4dfd689e43eef7fe46e516faf4.png is the probability *pi* of winning in casino *i*.

The next *q* lines each contain queries of one of the types specified above (1 ≤ *a* < *b* ≤ 109, 1 ≤ *i* ≤ *n*, 1 ≤ *l* ≤ *r* ≤ *n*).

It's guaranteed that there will be at least one query of type 2, i.e. the output will be non-empty. Additionally, it is guaranteed that *p* forms a non-decreasing sequence at all times.

**Output**

Print a real number for every request of type 2 — the probability that boy will "dominate" on that interval. Your answer will be considered correct if its absolute error does not exceed 10- 4.

Namely: let's assume that one of your answers is *a*, and the corresponding answer of the jury is *b*. The checker program will consider your answer correct if |*a* - *b*| ≤ 10- 4.

**Example**

**input**

3 13  
1 3  
1 2  
2 3  
2 1 1  
2 1 2  
2 1 3  
2 2 2  
2 2 3  
2 3 3  
1 2 2 3  
2 1 1  
2 1 2  
2 1 3  
2 2 2  
2 2 3  
2 3 3

**output**

0.3333333333  
0.2000000000  
0.1666666667  
0.5000000000  
0.4000000000  
0.6666666667  
0.3333333333  
0.2500000000  
0.2222222222  
0.6666666667  
0.5714285714  
0.6666666667

D. Memory and Scores

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Memory and his friend Lexa are competing to get higher score in one popular computer game. Memory starts with score *a* and Lexa starts with score *b*. In a single turn, both Memory and Lexa get some integer in the range [ - *k*;*k*] (i.e. one integer among  - *k*,  - *k* + 1,  - *k* + 2, ...,  - 2,  - 1, 0, 1, 2, ..., *k* - 1, *k*) and add them to their current scores. The game has exactly *t* turns. Memory and Lexa, however, are not good at this game, so they both always get a random integer at their turn.

Memory wonders how many possible games exist such that he ends with a strictly higher score than Lexa. Two games are considered to be different if in at least one turn at least one player gets different score. There are (2*k* + 1)2*t* games in total. Since the answer can be very large, you should print it modulo 109 + 7. Please solve this problem for Memory.

**Input**

The first and only line of input contains the four integers *a*, *b*, *k*, and *t* (1 ≤ *a*, *b* ≤ 100, 1 ≤ *k* ≤ 1000, 1 ≤ *t* ≤ 100) — the amount Memory and Lexa start with, the number *k*, and the number of turns respectively.

**Output**

Print the number of possible games satisfying the conditions modulo 1 000 000 007 (109 + 7) in one line.

**Examples**

**input**

1 2 2 1

**output**

6

**input**

1 1 1 2

**output**

31

**input**

2 12 3 1

**output**

0

**Note**

In the first sample test, Memory starts with 1 and Lexa starts with 2. If Lexa picks  - 2, Memory can pick 0, 1, or 2 to win. If Lexa picks  - 1, Memory can pick 1 or 2 to win. If Lexa picks 0, Memory can pick 2 to win. If Lexa picks 1 or 2, Memory cannot win. Thus, there are 3 + 2 + 1 = 6 possible games in which Memory wins.

C. Memory and De-Evolution

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Memory is now interested in the de-evolution of objects, specifically triangles. He starts with an equilateral triangle of side length *x*, and he wishes to perform operations to obtain an equilateral triangle of side length *y*.

In a single second, he can modify the length of a single side of the current triangle such that it remains a non-degenerate triangle (triangle of positive area). At any moment of time, the length of each side should be integer.

What is the minimum number of seconds required for Memory to obtain the equilateral triangle of side length *y*?

**Input**

The first and only line contains two integers *x* and *y* (3 ≤ *y* < *x* ≤ 100 000) — the starting and ending equilateral triangle side lengths respectively.

**Output**

Print a single integer — the minimum number of seconds required for Memory to obtain the equilateral triangle of side length *y* if he starts with the equilateral triangle of side length *x*.

**Examples**

**input**

6 3

**output**

4

**input**

8 5

**output**

3

**input**

22 4

**output**

6

**Note**

In the first sample test, Memory starts with an equilateral triangle of side length 6 and wants one of side length 3. Denote a triangle with sides *a*, *b*, and *c* as (*a*, *b*, *c*). Then, Memory can do http://codeforces.com/predownloaded/78/23/7823c31c27139a382b61aee582fd451f8c07905b.png.

In the second sample test, Memory can do http://codeforces.com/predownloaded/22/b2/22b2088d2e869848a78af18576dbfab0233bdbc2.png.

In the third sample test, Memory can do: http://codeforces.com/predownloaded/59/f9/59f9f1033e37dcff494a7de4ca5155e95e34da43.png

http://codeforces.com/predownloaded/7b/1a/7b1a1db6a7ff1635c9d24429cfe7208048f6d0f2.png.

B. Memory and Trident

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Memory is performing a walk on the two-dimensional plane, starting at the origin. He is given a string *s* with his directions for motion:

* An 'L' indicates he should move one unit left.
* An 'R' indicates he should move one unit right.
* A 'U' indicates he should move one unit up.
* A 'D' indicates he should move one unit down.

But now Memory wants to end at the origin. To do this, he has a special trident. This trident can replace any character in *s* with any of 'L', 'R', 'U', or 'D'. However, because he doesn't want to wear out the trident, he wants to make the minimum number of edits possible. Please tell Memory what is the minimum number of changes he needs to make to produce a string that, when walked, will end at the origin, or if there is no such string.

**Input**

The first and only line contains the string *s* (1 ≤ |*s*| ≤ 100 000) — the instructions Memory is given.

**Output**

If there is a string satisfying the conditions, output a single integer — the minimum number of edits required. In case it's not possible to change the sequence in such a way that it will bring Memory to to the origin, output -1.

**Examples**

**input**

RRU

**output**

-1

**input**

UDUR

**output**

1

**input**

RUUR

**output**

2

**Note**

In the first sample test, Memory is told to walk right, then right, then up. It is easy to see that it is impossible to edit these instructions to form a valid walk.

In the second sample test, Memory is told to walk up, then down, then up, then right. One possible solution is to change *s* to "LDUR". This string uses 1 edit, which is the minimum possible. It also ends at the origin.

A. Memory and Crow

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

There are *n* integers *b*1, *b*2, ..., *bn* written in a row. For all *i* from 1 to *n*, values *ai* are defined by the crows performing the following procedure:

* The crow sets *ai* initially 0.
* The crow then adds *bi* to *ai*, subtracts *bi*+ 1, adds the *bi*+ 2 number, and so on until the *n*'th number. Thus, *ai* = *bi* - *bi*+ 1 + *bi*+ 2 - *bi*+ 3....

Memory gives you the values *a*1, *a*2, ..., *an*, and he now wants you to find the initial numbers *b*1, *b*2, ..., *bn* written in the row? Can you do it?

**Input**

The first line of the input contains a single integer *n* (2 ≤ *n* ≤ 100 000) — the number of integers written in the row.

The next line contains *n*, the *i*'th of which is *ai* ( - 109 ≤ *ai* ≤ 109) — the value of the *i*'th number.

**Output**

Print *n* integers corresponding to the sequence *b*1, *b*2, ..., *bn*. It's guaranteed that the answer is unique and fits in 32-bit integer type.

**Examples**

**input**

5  
6 -4 8 -2 3

**output**

2 4 6 1 3

**input**

5  
3 -2 -1 5 6

**output**

1 -3 4 11 6

**Note**

In the first sample test, the crows report the numbers 6, - 4, 8, - 2, and 3 when he starts at indices 1, 2, 3, 4 and 5 respectively. It is easy to check that the sequence 2 4 6 1 3 satisfies the reports. For example, 6 = 2 - 4 + 6 - 1 + 3, and  - 4 = 4 - 6 + 1 - 3.

In the second sample test, the sequence 1,  - 3, 4, 11, 6 satisfies the reports. For example, 5 = 11 - 6 and 6 = 6.

E. ZS and The Birthday Paradox

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder has recently found an interesting concept called the Birthday Paradox. It states that given a random set of 23 people, there is around 50% chance that some two of them share the same birthday. ZS the Coder finds this very interesting, and decides to test this with the inhabitants of Udayland.

In Udayland, there are 2*n* days in a year. ZS the Coder wants to interview *k* people from Udayland, each of them has birthday in one of 2*n* days (each day with equal probability). He is interested in the probability of at least two of them have the birthday at the same day.

ZS the Coder knows that the answer can be written as an irreducible fraction http://codeforces.com/predownloaded/f6/5a/f65a04cbc303b96bef980005e4c8a396d71433e9.png. He wants to find the values of *A* and *B* (he does not like to deal with floating point numbers). Can you help him?

**Input**

The first and only line of the input contains two integers *n* and *k* (1 ≤ *n* ≤ 1018, 2 ≤ *k* ≤ 1018), meaning that there are 2*n* days in a year and that ZS the Coder wants to interview exactly *k* people.

**Output**

If the probability of at least two *k* people having the same birthday in 2*n* days long year equals http://codeforces.com/predownloaded/f6/5a/f65a04cbc303b96bef980005e4c8a396d71433e9.png (*A* ≥ 0, *B* ≥ 1, http://codeforces.com/predownloaded/48/ba/48ba4c75d4faed93cd3cfb4379c62a6f9d7eda08.png), print the *A* and *B* in a single line.

Since these numbers may be too large, print them modulo 106 + 3. Note that *A* and *B* must be coprime **before** their remainders modulo 106 + 3 are taken.

**Examples**

**input**

3 2

**output**

1 8

**input**

1 3

**output**

1 1

**input**

4 3

**output**

23 128

**Note**

In the first sample case, there are 23 = 8 days in Udayland. The probability that 2 people have the same birthday among 2 people is clearly http://codeforces.com/predownloaded/38/8d/388d6e4e353e0ecbf223f59095bd113e721b68a3.png, so *A* = 1, *B* = 8.

In the second sample case, there are only 21 = 2 days in Udayland, but there are 3 people, so it is guaranteed that two of them have the same birthday. Thus, the probability is 1 and *A* = *B* = 1.

D. Directed Roads

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder and Chris the Baboon has explored Udayland for quite some time. They realize that it consists of *n* towns numbered from 1 to *n*.

There are *n* directed roads in the Udayland. *i*-th of them goes from town *i* to some other town *ai* (*ai* ≠ *i*). ZS the Coder can flip the direction of any road in Udayland, i.e. if it goes from town *A* to town *B* before the flip, it will go from town *B* to town *A* after.

ZS the Coder considers the roads in the Udayland *confusing*, if there is a sequence of distinct towns *A*1, *A*2, ..., *Ak* (*k* > 1) such that for every 1 ≤ *i* < *k* there is a road from town *Ai* to town *Ai*+ 1 and another road from town *Ak* to town *A*1. In other words, the roads are confusing if **some of them** form a directed cycle of some towns.

Now ZS the Coder wonders how many sets of roads (there are 2*n* variants) in initial configuration can he choose to flip such that after flipping each road in the set exactly once, the resulting network will **not** be confusing.

Note that it is allowed that after the flipping there are more than one directed road from some town and possibly some towns with no roads leading out of it, or multiple roads between any pair of cities.

**Input**

The first line of the input contains single integer *n* (2 ≤ *n* ≤ 2·105) — the number of towns in Udayland.

The next line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ *n*, *ai* ≠ *i*), *ai* denotes a road going from town *i* to town *ai*.

**Output**

Print a single integer — the number of ways to flip some set of the roads so that the resulting whole set of all roads is not confusing. Since this number may be too large, print the answer modulo 109 + 7.

**Examples**

**input**

3  
2 3 1

**output**

6

**input**

4  
2 1 1 1

**output**

8

**input**

5  
2 4 2 5 3

**output**

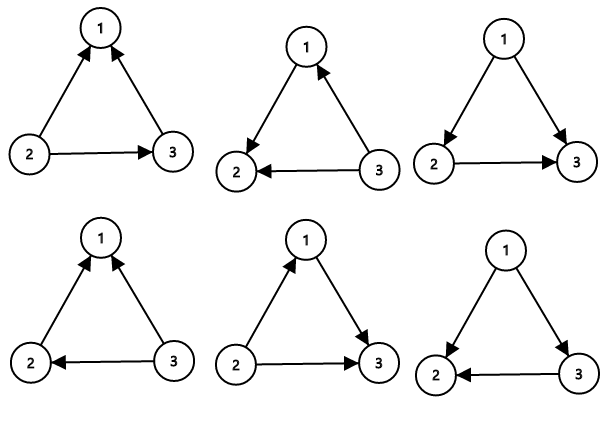
28

**Note**

Consider the first sample case. There are 3 towns and 3 roads. The towns are numbered from 1 to 3 and the roads are http://codeforces.com/predownloaded/ab/3a/ab3aa0026cb5ab74ad3902d3590f907b613f69c4.png, http://codeforces.com/predownloaded/87/b9/87b9bfe02d6490736188a0be16d7fef0a4279045.png, http://codeforces.com/predownloaded/11/40/1140548295e0c0bf64e12e6f56488dfe92f20fd4.png initially. Number the roads 1 to 3 in this order.

The sets of roads that ZS the Coder can flip (to make them not confusing) are {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}. Note that the empty set is invalid because if no roads are flipped, then towns 1, 2, 3 is form a directed cycle, so it is confusing. Similarly, flipping all roads is confusing too. Thus, there are a total of 6 possible sets ZS the Coder can flip.

The sample image shows all possible ways of orienting the roads from the first sample such that the network is **not confusing**.



C. Coloring Trees

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder and Chris the Baboon has arrived at Udayland! They walked in the park where *n* trees grow. They decided to be naughty and color the trees in the park. The trees are numbered with integers from 1 to *n* from left to right.

Initially, tree *i* has color *ci*. ZS the Coder and Chris the Baboon recognizes only *m* different colors, so 0 ≤ *ci* ≤ *m*, where *ci* = 0 means that tree *i* is *uncolored*.

ZS the Coder and Chris the Baboon decides to color only the uncolored trees, i.e. the trees with *ci* = 0. They can color each of them them in any of the *m* colors from 1 to *m*. Coloring the *i*-th tree with color *j* requires exactly *pi*,*j* litres of paint.

The two friends define the *beauty* of a coloring of the trees as the **minimum** number of contiguous groups (each group contains some subsegment of trees) you can split all the *n* trees into so that each group contains trees of the same color. For example, if the colors of the trees from left to right are 2, 1, 1, 1, 3, 2, 2, 3, 1, 3, the beauty of the coloring is 7, since we can partition the trees into 7 contiguous groups of the same color : {2}, {1, 1, 1}, {3}, {2, 2}, {3}, {1}, {3}.

ZS the Coder and Chris the Baboon wants to color all uncolored trees so that the beauty of the coloring is **exactly** *k*. They need your help to determine the minimum amount of paint (in litres) needed to finish the job.

Please note that the friends can't color the trees that are already colored.

**Input**

The first line contains three integers, *n*, *m* and *k* (1 ≤ *k* ≤ *n* ≤ 100, 1 ≤ *m* ≤ 100) — the number of trees, number of colors and beauty of the resulting coloring respectively.

The second line contains *n* integers *c*1, *c*2, ..., *cn* (0 ≤ *ci* ≤ *m*), the initial colors of the trees. *ci* equals to 0 if the tree number *i* is uncolored, otherwise the *i*-th tree has color *ci*.

Then *n* lines follow. Each of them contains *m* integers. The *j*-th number on the *i*-th of them line denotes *pi*,*j* (1 ≤ *pi*,*j* ≤ 109) — the amount of litres the friends need to color *i*-th tree with color *j*. *pi*,*j*'s are specified even for the initially colored trees, but such trees still can't be colored.

**Output**

Print a single integer, the minimum amount of paint needed to color the trees. If there are no valid tree colorings of beauty *k*, print  - 1.

**Examples**

**input**

3 2 2  
0 0 0  
1 2  
3 4  
5 6

**output**

10

**input**

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

**output**

-1

**input**

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

**output**

5

**input**

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

**output**

0

**Note**

In the first sample case, coloring the trees with colors 2, 1, 1 minimizes the amount of paint used, which equals to 2 + 3 + 5 = 10. Note that 1, 1, 1 would not be valid because the beauty of such coloring equals to 1 ({1, 1, 1} is a way to group the trees into a single group of the same color).

In the second sample case, all the trees are colored, but the beauty of the coloring is 3, so there is no valid coloring, and the answer is  - 1.

In the last sample case, all the trees are colored and the beauty of the coloring matches *k*, so no paint is used and the answer is 0.

B. Chris and Magic Square

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder and Chris the Baboon arrived at the entrance of Udayland. There is a *n* × *n* magic grid on the entrance which is filled with integers. Chris noticed that exactly one of the cells in the grid is empty, and to enter Udayland, they need to fill a **positive integer** into the empty cell.

Chris tried filling in random numbers but it didn't work. ZS the Coder realizes that they need to fill in a positive integer such that the numbers in the grid form *a magic square*. This means that he has to fill in a positive integer so that the sum of the numbers in each row of the grid (http://codeforces.com/predownloaded/e0/de/e0dedf4f5ae215df380bdb20bb1b2f792d1c1229.png), each column of the grid (http://codeforces.com/predownloaded/f6/c2/f6c297cd539326366bbd52d96bc948094081a192.png), and the two long diagonals of the grid (the main diagonal — http://codeforces.com/predownloaded/e4/6b/e46b9f425e1ff0af3b28056859326374930795aa.png and the secondary diagonal — http://codeforces.com/predownloaded/a8/27/a827538213ceb8bffba39fc2010c7bddc2d79fc6.png) are equal.

Chris doesn't know what number to fill in. Can you help Chris find the correct positive integer to fill in or determine that it is impossible?

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 500) — the number of rows and columns of the magic grid.

*n* lines follow, each of them contains *n* integers. The *j*-th number in the *i*-th of them denotes *ai*,*j* (1 ≤ *ai*,*j* ≤ 109 or *ai*,*j* = 0), the number in the *i*-th row and *j*-th column of the magic grid. If the corresponding cell is empty, *ai*,*j* will be equal to 0. Otherwise, *ai*,*j* is **positive**.

It is guaranteed that there is exactly one pair of integers *i*, *j* (1 ≤ *i*, *j* ≤ *n*) such that *ai*,*j* = 0.

**Output**

Output a single integer, the positive integer *x* (1 ≤ *x* ≤ 1018) that should be filled in the empty cell so that the whole grid becomes a magic square. If such positive integer *x* does not exist, output  - 1 instead.

If there are multiple solutions, you may print any of them.

**Examples**

**input**

3  
4 0 2  
3 5 7  
8 1 6

**output**

9

**input**

4  
1 1 1 1  
1 1 0 1  
1 1 1 1  
1 1 1 1

**output**

1

**input**

4  
1 1 1 1  
1 1 0 1  
1 1 2 1  
1 1 1 1

**output**

-1

**Note**

In the first sample case, we can fill in 9 into the empty cell to make the resulting grid a magic square. Indeed,

The sum of numbers in each row is:

4 + 9 + 2 = 3 + 5 + 7 = 8 + 1 + 6 = 15.

The sum of numbers in each column is:

4 + 3 + 8 = 9 + 5 + 1 = 2 + 7 + 6 = 15.

The sum of numbers in the two diagonals is:

4 + 5 + 6 = 2 + 5 + 8 = 15.

In the third sample case, it is impossible to fill a number in the empty square such that the resulting grid is a magic square.

A. Bus to Udayland

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

ZS the Coder and Chris the Baboon are travelling to Udayland! To get there, they have to get on the special IOI bus. The IOI bus has *n*rows of seats. There are 4 seats in each row, and the seats are separated into pairs by a walkway. When ZS and Chris came, some places in the bus was already occupied.

ZS and Chris are good friends. They insist to get **a pair of neighbouring empty seats**. Two seats are considered neighbouring if they are in the same row and in the same pair. Given the configuration of the bus, can you help ZS and Chris determine where they should sit?

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 1000) — the number of rows of seats in the bus.

Then, *n* lines follow. Each line contains exactly 5 characters, the first two of them denote the first pair of seats in the row, the third character denotes the walkway (it always equals '|') and the last two of them denote the second pair of seats in the row.

Each character, except the walkway, equals to 'O' or to 'X'. 'O' denotes an empty seat, 'X' denotes an occupied seat. See the sample cases for more details.

**Output**

If it is possible for Chris and ZS to sit at neighbouring empty seats, print "YES" (without quotes) in the first line. In the next *n* lines print the bus configuration, where the characters in the pair of seats for Chris and ZS is changed with characters '+'. Thus the configuration should differ from the input one by exactly two charaters (they should be equal to 'O' in the input and to '+' in the output).

If there is no pair of seats for Chris and ZS, print "NO" (without quotes) in a single line.

If there are multiple solutions, you may print any of them.

**Examples**

**input**

6  
OO|OX  
XO|XX  
OX|OO  
XX|OX  
OO|OO  
OO|XX

**output**

YES  
++|OX  
XO|XX  
OX|OO  
XX|OX  
OO|OO  
OO|XX

**input**

4  
XO|OX  
XO|XX  
OX|OX  
XX|OX

**output**

NO

**input**

5  
XX|XX  
XX|XX  
XO|OX  
XO|OO  
OX|XO

**output**

YES  
XX|XX  
XX|XX  
XO|OX  
XO|++  
OX|XO

**Note**

Note that the following is an incorrect configuration for the first sample case because the seats must be in the same pair.

O+|+X

XO|XX

OX|OO

XX|OX

OO|OO

OO|XX

F. String Set Queries

time limit per test

3 seconds

memory limit per test

768 megabytes

input

standard input

output

standard output

You should process *m* queries over a set *D* of strings. Each query is one of three kinds:

1. Add a string *s* to the set *D*. It is guaranteed that the string *s* was not added before.
2. Delete a string *s* from the set *D*. It is guaranteed that the string *s* is in the set *D*.
3. For the given string *s* find the number of occurrences of the strings from the set *D*. If some string *p* from *D* has several occurrences in *s* you should count all of them.

Note that you should solve the problem in online mode. It means that you can't read the whole input at once. You can read each query only after writing the answer for the last query of the third type. Use functions fflush in C++ and BufferedWriter.flush in Javalanguages after each writing in your program.

**Input**

The first line contains integer *m* (1 ≤ *m* ≤ 3·105) — the number of queries.

Each of the next *m* lines contains integer *t* (1 ≤ *t* ≤ 3) and nonempty string *s* — the kind of the query and the string to process. All strings consist of only lowercase English letters.

The sum of lengths of all strings in the input will not exceed 3·105.

**Output**

For each query of the third kind print the only integer *c* — the desired number of occurrences in the string *s*.

**Examples**

**input**

5  
1 abc  
3 abcabc  
2 abc  
1 aba  
3 abababc

**output**

2  
2

**input**

10  
1 abc  
1 bcd  
1 abcd  
3 abcd  
2 abcd  
3 abcd  
2 bcd  
3 abcd  
2 abc  
3 abcd

**output**

3  
2  
1  
0

E. Generate a String

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

zscoder wants to generate an input file for some programming competition problem.

His input is a string consisting of *n* letters 'a'. He is too lazy to write a generator so he will manually generate the input in a text editor.

Initially, the text editor is empty. It takes him *x* seconds to insert or delete a letter 'a' from the text file and *y* seconds to copy the contents of the entire text file, and duplicate it.

zscoder wants to find the minimum amount of time needed for him to create the input file of exactly *n* letters 'a'. Help him to determine the amount of time needed to generate the input.

**Input**

The only line contains three integers *n*, *x* and *y* (1 ≤ *n* ≤ 107, 1 ≤ *x*, *y* ≤ 109) — the number of letters 'a' in the input file and the parameters from the problem statement.

**Output**

Print the only integer *t* — the minimum amount of time needed to generate the input file.

**Examples**

**input**

8 1 1

**output**

4

**input**

8 1 10

**output**

8

D. Two Arithmetic Progressions

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given two arithmetic progressions: *a*1*k* + *b*1 and *a*2*l* + *b*2. Find the number of integers *x* such that *L* ≤ *x* ≤ *R* and *x* = *a*1*k*' + *b*1 = *a*2*l*' + *b*2, for some integers *k*', *l*' ≥ 0.

**Input**

The only line contains six integers *a*1, *b*1, *a*2, *b*2, *L*, *R* (0 < *a*1, *a*2 ≤ 2·109,  - 2·109 ≤ *b*1, *b*2, *L*, *R* ≤ 2·109, *L* ≤ *R*).

**Output**

Print the desired number of integers *x*.

**Examples**

**input**

2 0 3 3 5 21

**output**

3

**input**

2 4 3 0 6 17

**output**

2

C. Magic Odd Square

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Find an *n* × *n* matrix with different numbers from 1 to *n*2, so the sum in each row, column and both main diagonals are odd.

**Input**

The only line contains odd integer *n* (1 ≤ *n* ≤ 49).

**Output**

Print *n* lines with *n* integers. All the integers should be different and from 1 to *n*2. The sum in each row, column and both main diagonals should be odd.

**Examples**

**input**

1

**output**

1

**input**

3

**output**

2 1 4  
3 5 7  
6 9 8

B. Optimal Point on a Line

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given *n* points on a line with their coordinates *xi*. Find the point *x* so the sum of distances to the given points is minimal.

**Input**

The first line contains integer *n* (1 ≤ *n* ≤ 3·105) — the number of points on the line.

The second line contains *n* integers *xi* ( - 109 ≤ *xi* ≤ 109) — the coordinates of the given *n* points.

**Output**

Print the only integer *x* — the position of the optimal point on the line. If there are several optimal points print the position of the leftmost one. It is guaranteed that the answer is always the integer.

**Example**

**input**

4  
1 2 3 4

**output**

2

A. King Moves

time limit per test

1 second

memory limit per test

256 megabytes

input

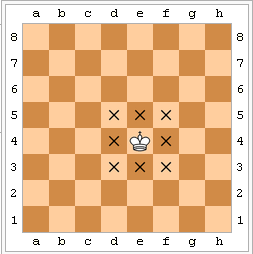
standard input

output

standard output

The only king stands on the standard chess board. You are given his position in format "cd", where *c* is the column from 'a' to 'h' and *d*is the row from '1' to '8'. Find the number of moves permitted for the king.

Check the king's moves here https://en.wikipedia.org/wiki/King\_(chess).

King moves from the position e4

**Input**

The only line contains the king's position in the format "cd", where 'c' is the column from 'a' to 'h' and 'd' is the row from '1' to '8'.

**Output**

Print the only integer *x* — the number of moves permitted for the king.

**Example**

**input**

e4

**output**

8

B. Checkpoints

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasya takes part in the orienteering competition. There are *n* checkpoints located along the line at coordinates *x*1, *x*2, ..., *xn*. Vasya starts at the point with coordinate *a*. His goal is to visit at least *n* - 1 checkpoint in order to finish the competition. Participant are allowed to visit checkpoints in arbitrary order.

Vasya wants to pick such checkpoints and the order of visiting them that the total distance travelled is minimized. He asks you to calculate this minimum possible value.

**Input**

The first line of the input contains two integers *n* and *a* (1 ≤ *n* ≤ 100 000,  - 1 000 000 ≤ *a* ≤ 1 000 000) — the number of checkpoints and Vasya's starting position respectively.

The second line contains *n* integers *x*1, *x*2, ..., *xn* ( - 1 000 000 ≤ *xi* ≤ 1 000 000) — coordinates of the checkpoints.

**Output**

Print one integer — the minimum distance Vasya has to travel in order to visit at least *n* - 1 checkpoint.

**Examples**

**input**

3 10  
1 7 12

**output**

7

**input**

2 0  
11 -10

**output**

10

**input**

5 0  
0 0 1000 0 0

**output**

0

**Note**

In the first sample Vasya has to visit at least two checkpoints. The optimal way to achieve this is the walk to the third checkpoints (distance is 12 - 10 = 2) and then proceed to the second one (distance is 12 - 7 = 5). The total distance is equal to 2 + 5 = 7.

In the second sample it's enough to visit only one checkpoint so Vasya should just walk to the point  - 10.

A. Juicer

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Kolya is going to make fresh orange juice. He has *n* oranges of sizes *a*1, *a*2, ..., *an*. Kolya will put them in the juicer in the fixed order, starting with orange of size *a*1, then orange of size *a*2 and so on. To be put in the juicer the orange must have size not exceeding *b*, so if Kolya sees an orange that is strictly greater he throws it away and continues with the next one.

The juicer has a special section to collect waste. It overflows if Kolya squeezes oranges of the total size strictly greater than *d*. When it happens Kolya empties the waste section (even if there are no more oranges) and continues to squeeze the juice. How many times will he have to empty the waste section?

**Input**

The first line of the input contains three integers *n*, *b* and *d* (1 ≤ *n* ≤ 100 000, 1 ≤ *b* ≤ *d* ≤ 1 000 000) — the number of oranges, the maximum size of the orange that fits in the juicer and the value *d*, which determines the condition when the waste section should be emptied.

The second line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 1 000 000) — sizes of the oranges listed in the order Kolya is going to try to put them in the juicer.

**Output**

Print one integer — the number of times Kolya will have to empty the waste section.

**Examples**

**input**

2 7 10  
5 6

**output**

1

**input**

1 5 10  
7

**output**

0

**input**

3 10 10  
5 7 7

**output**

1

**input**

1 1 1  
1

**output**

0

**Note**

In the first sample, Kolya will squeeze the juice from two oranges and empty the waste section afterwards.

In the second sample, the orange won't fit in the juicer so Kolya will have no juice at all.

E. Student's Camp

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Alex studied well and won the trip to student camp Alushta, located on the seashore.

Unfortunately, it's the period of the strong winds now and there is a chance the camp will be destroyed! Camp building can be represented as the rectangle of *n* + 2 concrete blocks height and *m* blocks width.

Every day there is a breeze blowing from the sea. Each block, except for the blocks of the upper and lower levers, such that there is no block to the left of it is destroyed with the probability http://codeforces.com/predownloaded/ae/bc/aebc9a1718343b8d2aa26810f09cf68919ffdf45.png. Similarly, each night the breeze blows in the direction to the sea. Thus, each block (again, except for the blocks of the upper and lower levers) such that there is no block to the right of it is destroyed with the same probability *p*. Note, that blocks of the upper and lower level are **indestructible**, so there are only *n*·*m* blocks that can be destroyed.

The period of the strong winds will last for *k* days and *k* nights. If during this period the building will split in at least two connected components, it will collapse and Alex will have to find another place to spend summer.

Find the probability that Alex won't have to look for other opportunities and will be able to spend the summer in this camp.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 1500) that define the size of the destructible part of building.

The second line of the input contains two integers *a* and *b* (1 ≤ *a* ≤ *b* ≤ 109) that define the probability *p*. It's guaranteed that integers *a*and *b* are coprime.

The third line contains a single integer *k* (0 ≤ *k* ≤ 100 000) — the number of days and nights strong wind will blow for.

**Output**

Consider the answer as an irreducible fraction is equal to http://codeforces.com/predownloaded/79/0d/790dac5b2fbfd2e79476cfb1b7cba21944d25b29.png. Print one integer equal to http://codeforces.com/predownloaded/e3/05/e3059d56f4a6faa31e533ecc5d3b20d6fe191ca2.png. It's guaranteed that within the given constraints http://codeforces.com/predownloaded/b8/fa/b8fa962e83dae20b311c949e2aee7282413a58ca.png.

**Examples**

**input**

2 2  
1 2  
1

**output**

937500007

**input**

5 1  
3 10  
1

**output**

95964640

**input**

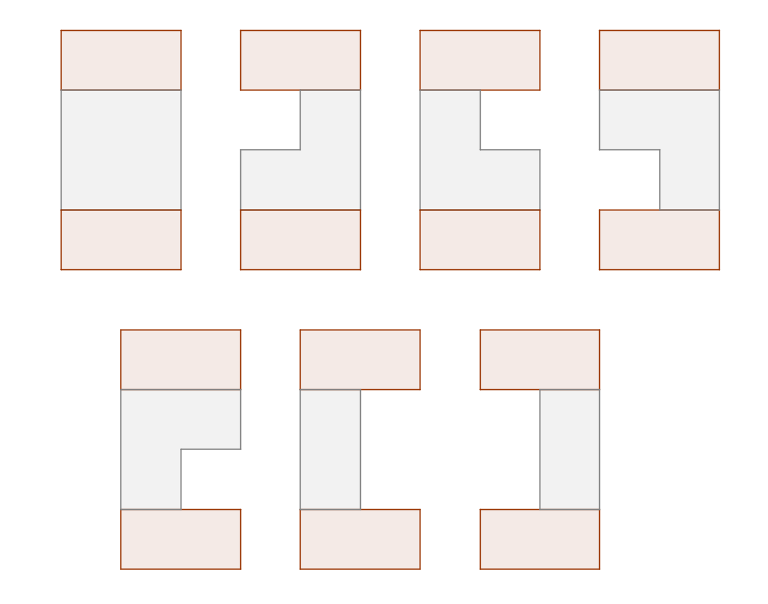
3 3  
1 10  
5

**output**

927188454

**Note**

In the first sample, each of the four blocks is destroyed with the probability http://codeforces.com/predownloaded/06/47/06479cd23d60c92802b763b2307abea3d29db615.png. There are 7 scenarios that result in building not collapsing, and the probability we are looking for is equal to http://codeforces.com/predownloaded/5a/c1/5ac14015e547cfa6c97c973a008e7051c1b3fd25.png, so you should print http://codeforces.com/predownloaded/11/bf/11bf9687ebbdaf4b4b26d78d44c03de52bf26e51.png



D. Incorrect Flow

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

At the entrance examination for the magistracy of the MSU Cyber-Mechanics Department Sasha got the question about Ford-Fulkerson algorithm. He knew the topic perfectly as he worked with it many times on programming competition. As the task for the question he was given a network with partially build flow that he had to use in order to demonstrate the workflow of the algorithm. He quickly finished to write the text and took a look at the problem only to understand that the given network is incorrect!

Suppose you are given a directed graph *G*(*V*, *E*) with two special nodes *s* and *t* called source and sink. We denote as *n* the number of nodes in the graph, i.e. *n* = |*V*| and *m* stands for the number of directed edges in the graph, i.e. *m* = |*E*|. For the purpose of this problem we always consider node 1 to be the source and node *n* to be the sink. In addition, for each edge of the graph *e* we define the capacity function *c*(*e*) and flow function *f*(*e*). Function *f*(*e*) represents the correct flow if the following conditions are satisfied:

1. For each edge http://codeforces.com/predownloaded/ca/0b/ca0bc3567a1bbaf42ef8632cc27c9ccf5db3ca59.png the flow is non-negative and does not exceed capacity *c*(*e*), i.e. 0 ≤ *f*(*e*) ≤ *c*(*e*).
2. For each node http://codeforces.com/predownloaded/46/5e/465e068498b4267ce8c2bb90ef74c0ed932fe3c0.png, that is not source or sink (*v* ≠ *s* and *v* ≠ *t*) the sum of flows of all edges going in *v* is equal to the sum of the flows among all edges going out from *v*. In other words, there is no flow stuck in *v*.

It was clear that as the exam was prepared last night and there are plenty of mistakes in the tasks. Sasha asked one of the professors to fix the network or give the correct task, but the reply was that the magistrate student should be able to fix the network himself. As the professor doesn't want the task to become easier, he asks Sasha to fix the network in a such way that the total number of changes is minimum possible. Sasha is not allowed to remove edges, add new ones or reverse the direction of existing edges. The only thing he is able to do is to change capacity function *c*(*e*) and flow function *f*(*e*). Moreover, all the values should remain non-negative integers. There is no requirement on the flow to be maximum in any sense.

Find the minimum possible total change of the functions *f*(*e*) and *c*(*e*) that Sasha has to make in order to make the flow correct. The total change is defined as the sum of absolute differences, i.e. if new functions are *f*\*(*e*) and *c*\*(*e*), then the total change is http://codeforces.com/predownloaded/b7/ed/b7ed4dd0e0aec35d42a9348b1a8cf9688d1d5a5d.png.

**Input**

The first line of the input contains two integers *n* and *m* (2 ≤ *n* ≤ 100, 0 ≤ *m* ≤ 100) — the number of nodes and edges in the graph respectively. Each of the following *m* lines contains the description of the edges, consisting of four integers *ui*, *vi*, *ci* and *fi* (1 ≤ *ui*, *vi* ≤ *n*, *ui* ≠ *vi*, 0 ≤ *ci*, *fi* ≤ 1 000 000) — index of the node the edges starts from, the index of the node the edge goes to, current capacity and flow value.

Node number 1 is the source, and node number *n* is the sink. It's guaranteed that no edge goes to the source, and no edges starts in the sink.

Given graph contains no self-loops but may contain multiple edges.

**Output**

Print one integer — the minimum total sum of changes that Sasha has to do in order to get the correct flow description.

**Examples**

**input**

2 1  
1 2 2 1

**output**

0

**input**

2 1  
1 2 1 2

**output**

1

**input**

3 3  
1 2 1 1  
2 3 2 2  
1 3 3 3

**output**

1

**input**

4 2  
2 3 1 1  
3 2 1 1

**output**

0

**Note**

In the first sample, the flow is initially correct. Note, that the flow is not maximum, but this is not required.

In the second sample, the flow value of the only edge is greater than its capacity. There are two ways to fix this: either increase the capacity up to 2 or reduce the flow down to 1.

In the third sample, there is only 1 unit of flow coming to vertex 2, but there are 2 units going out of it. One of the possible solutions is to reduce the value of the flow on the second edge by 1.

In the fourth sample, there is isolated circulation of flow, but this description is correct by definition.

C. Centroids

time limit per test

4 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

*Tree* is a connected acyclic graph. Suppose you are given a tree consisting of *n* vertices. The vertex of this tree is called *centroid* if the size of each connected component that appears if this vertex is removed from the tree doesn't exceed http://codeforces.com/predownloaded/39/fd/39fdaa6f330f6afe718c8e14bc2d862f48999d59.png.

You are given a tree of size *n* and can perform no more than one edge replacement. *Edge replacement* is the operation of removing one edge from the tree (without deleting incident vertices) and inserting one new edge (without adding new vertices) in such a way that the graph remains a tree. For each vertex you have to determine if it's possible to make it centroid by performing no more than one edge replacement.

**Input**

The first line of the input contains an integer *n* (2 ≤ *n* ≤ 400 000) — the number of vertices in the tree. Each of the next *n* - 1 lines contains a pair of vertex indices *ui* and *vi* (1 ≤ *ui*, *vi* ≤ *n*) — endpoints of the corresponding edge.

**Output**

Print *n* integers. The *i*-th of them should be equal to 1 if the *i*-th vertex can be made centroid by replacing no more than one edge, and should be equal to 0 otherwise.

**Examples**

**input**

3  
1 2  
2 3

**output**

1 1 1

**input**

5  
1 2  
1 3  
1 4  
1 5

**output**

1 0 0 0 0

**Note**

In the first sample each vertex can be made a centroid. For example, in order to turn vertex 1 to centroid one have to replace the edge (2, 3) with the edge (1, 3).

B. Recover the String

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

For each string *s* consisting of characters '0' and '1' one can define four integers *a*00, *a*01, *a*10 and *a*11, where *axy* is the number of **subsequences** of length 2 of the string *s* equal to the sequence {*x*, *y*}.

In these problem you are given four integers *a*00, *a*01, *a*10, *a*11 and have to find any non-empty string *s* that matches them, or determine that there is no such string. One can prove that if at least one answer exists, there exists an answer of length no more than 1 000 000.

**Input**

The only line of the input contains four non-negative integers *a*00, *a*01, *a*10 and *a*11. Each of them doesn't exceed 109.

**Output**

If there exists a non-empty string that matches four integers from the input, print it in the only line of the output. Otherwise, print "Impossible". The length of your answer must not exceed 1 000 000.

**Examples**

**input**

1 2 3 4

**output**

Impossible

**input**

1 2 2 1

**output**

0110

A. Letters Cyclic Shift

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given a non-empty string *s* consisting of lowercase English letters. You have to pick **exactly one non-empty substring** of *s* and shift all its letters 'z' http://codeforces.com/predownloaded/9e/58/9e58594e343ae98e6885aaa282186965f4b9653b.png 'y' http://codeforces.com/predownloaded/9e/58/9e58594e343ae98e6885aaa282186965f4b9653b.png 'x' http://codeforces.com/predownloaded/1b/01/1b01be31aaf100be438d2b4934c93d5d1d32d790.png 'b' http://codeforces.com/predownloaded/9e/58/9e58594e343ae98e6885aaa282186965f4b9653b.png 'a' http://codeforces.com/predownloaded/9e/58/9e58594e343ae98e6885aaa282186965f4b9653b.png 'z'. In other words, each character is replaced with the previous character of English alphabet and 'a' is replaced with 'z'.

What is the lexicographically minimum string that can be obtained from *s* by performing this shift exactly once?

**Input**

The only line of the input contains the string *s* (1 ≤ |*s*| ≤ 100 000) consisting of lowercase English letters.

**Output**

Print the lexicographically minimum string that can be obtained from *s* by shifting letters of exactly one non-empty substring.

**Examples**

**input**

codeforces

**output**

bncdenqbdr

**input**

abacaba

**output**

aaacaba

**Note**

String *s* is lexicographically smaller than some other string *t* of the same length if there exists some 1 ≤ *i* ≤ |*s*|, such that *s*1 = *t*1, *s*2 = *t*2, ..., *si*- 1 = *ti*- 1, and *si* < *ti*.

E. Garlands

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

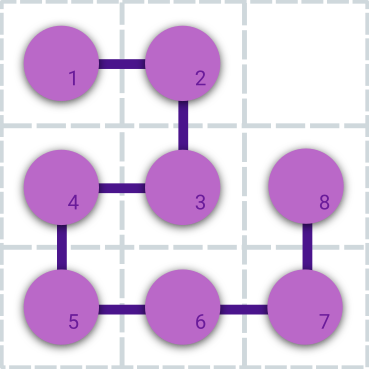
output

standard output

Like all children, Alesha loves New Year celebration. During the celebration he and his whole family dress up the fir-tree. Like all children, Alesha likes to play with garlands — chains consisting of a lightbulbs.

Alesha uses a grid field sized *n* × *m* for playing. The rows of the field are numbered from 1 to *n* from the top to the bottom and columns are numbered from 1 to *m* from the left to the right.

Alesha has *k* garlands which he places at the field. He does so in the way such that each lightbulb of each garland lies in the center of some cell in the field, and each cell contains **at most one lightbulb**. Of course lightbulbs, which are neighbours in some garland, appears in cells neighbouring by a side.



The example of garland placing.

Each garland is turned off or turned on at any moment. If some garland is turned on then each of its lightbulbs is turned on, the same applies for garland turned off. Each lightbulb in the whole garland set is unique, and thus, being turned on, brings Alesha some pleasure, described by an integer value. Turned off lightbulbs don't bring Alesha any pleasure.

Alesha can turn garlands on and off and wants to know the sum of pleasure value which the lightbulbs, placed in the centers of the cells in some rectangular part of the field, bring him. Initially **all the garlands are turned on**.

Alesha is still very little and can't add big numbers. He extremely asks you to help him.

**Input**

The first line of the input contains three integers *n*, *m* and *k* (1 ≤ *n*, *m*, *k* ≤ 2000) — the number of field rows, the number of field columns and the number of garlands placed at the field respectively.

Next lines contains garlands set description in the following format:

The first line of a single garland description contains a single integer *len* (1 ≤ *len* ≤ 2000) — the number of lightbulbs in the garland.

Each of the next *len* lines contains three integers *i*, *j* and *w* (1 ≤ *i* ≤ *n*, 1 ≤ *j* ≤ *m*, 1 ≤ *w* ≤ 109) — the coordinates of the cell containing a lightbullb and pleasure value Alesha gets from it if it is turned on. The lightbulbs are given in the order they are forming a chain in the garland. It is guaranteed that neighbouring lightbulbs are placed in the cells neighbouring by a side.

The next line contains single integer *q* (1 ≤ *q* ≤ 106) — the number of events in Alesha's game. The next *q* lines describes events in chronological order. The *i*-th of them describes the *i*-th event in the one of the following formats:

* SWITCH *i* — Alesha turns off *i*-th garland if it is turned on, or turns it on if it is turned off. It is guaranteed that 1 ≤ *i* ≤ *k*.
* ASK *x*1 *y*1 *x*2 *y*2 — Alesha wants to know the sum of pleasure values the lightbulbs, placed in a rectangular part of the field. Top-left cell of a part has coordinates (*x*1, *y*1) and right-bottom cell has coordinates (*x*2, *y*2). It is guaranteed that 1 ≤ *x*1 ≤ *x*2 ≤ *n* and 1 ≤ *y*1 ≤ *y*2 ≤ *m*. There is **no more than 2000** events of this type in the input.

All the numbers in the input are integers.

Please note that the input is quite large, so be careful while using some input ways. In particular, it's not recommended to use cin in codes on C++ and class Scanner in codes on Java.

**Output**

For each ASK operation print the sum Alesha wants to know in a separate line. Print the answers in chronological order.

**Examples**

**input**

4 4 3  
5  
1 1 2  
1 2 3  
2 2 1  
2 1 4  
3 1 7  
4  
1 3 1  
2 3 3  
2 4 3  
1 4 1  
7  
4 1 1  
4 2 9  
3 2 8  
3 3 3  
4 3 4  
4 4 1  
3 4 1  
2  
ASK 2 2 3 3  
ASK 1 1 4 4

**output**

15  
52

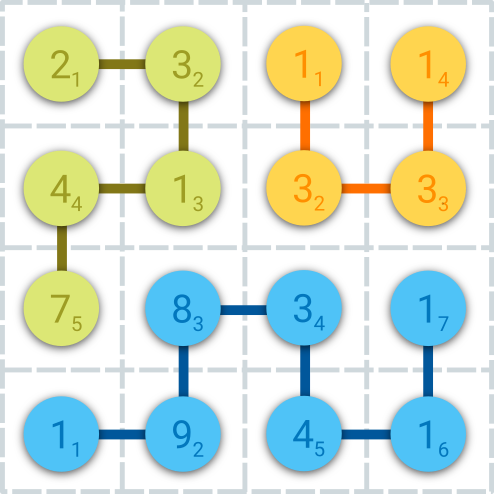
**input**

4 4 1  
8  
4 1 1  
3 1 2  
2 1 1  
1 1 7  
1 2 5  
2 2 4  
2 3 1  
1 3 1  
3  
ASK 1 1 3 2  
SWITCH 1  
ASK 1 1 3 2

**output**

19  
0

**Note**



This image illustrates the first sample case.

D. Persistent Bookcase

time limit per test

2 seconds

memory limit per test

512 megabytes

input

standard input

output

standard output

Recently in school Alina has learned what are the *persistent data structures*: they are data structures that always preserves the previous version of itself and access to it when it is modified.

After reaching home Alina decided to invent her own persistent data structure. Inventing didn't take long: there is a bookcase right behind her bed. Alina thinks that the bookcase is a good choice for a persistent data structure. Initially the bookcase is empty, thus there is no book at any position at any shelf.

The bookcase consists of *n* shelves, and each shelf has exactly *m* positions for books at it. Alina enumerates shelves by integers from 1to *n* and positions at shelves — from 1 to *m*. Initially the bookcase is empty, thus there is no book at any position at any shelf in it.

Alina wrote down *q* operations, which will be consecutively applied to the bookcase. Each of the operations has one of four types:

* 1 *i* *j* — Place a book at position *j* at shelf *i* if there is no book at it.
* 2 *i* *j* — Remove the book from position *j* at shelf *i* if there is a book at it.
* 3 *i* — Invert book placing at shelf *i*. This means that from every position at shelf *i* which has a book at it, the book should be removed, and at every position at shelf *i* which has not book at it, a book should be placed.
* 4 *k* — Return the books in the bookcase in a state they were after applying *k*-th operation. In particular, *k* = 0 means that the bookcase should be in initial state, thus every book in the bookcase should be removed from its position.

After applying each of operation Alina is interested in the number of books in the bookcase. Alina got 'A' in the school and had no problem finding this values. Will you do so?

**Input**

The first line of the input contains three integers *n*, *m* and *q* (1 ≤ *n*, *m* ≤ 103, 1 ≤ *q* ≤ 105) — the bookcase dimensions and the number of operations respectively.

The next *q* lines describes operations in chronological order — *i*-th of them describes *i*-th operation in one of the four formats described in the statement.

It is guaranteed that shelf indices and position indices are correct, and in each of fourth-type operation the number *k* corresponds to some operation before it or equals to 0.

**Output**

For each operation, print the number of books in the bookcase after applying it in a separate line. The answers should be printed in chronological order.

**Examples**

**input**

2 3 3  
1 1 1  
3 2  
4 0

**output**

1  
4  
0

**input**

4 2 6  
3 2  
2 2 2  
3 3  
3 2  
2 2 2  
3 2

**output**

2  
1  
3  
3  
2  
4

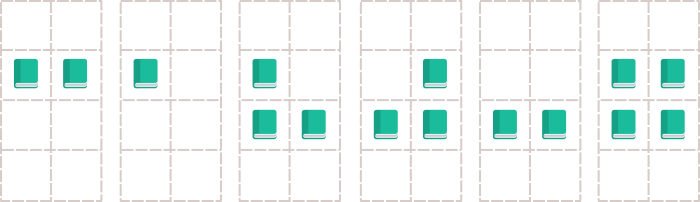
**input**

2 2 2  
3 2  
2 2 1

**output**

2  
1

**Note**



This image illustrates the second sample case.

C. Pythagorean Triples

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Katya studies in a fifth grade. Recently her class studied right triangles and the Pythagorean theorem. It appeared, that there are triples of positive integers such that you can construct a right triangle with segments of lengths corresponding to triple. Such triples are called *Pythagorean triples*.

For example, triples (3, 4, 5), (5, 12, 13) and (6, 8, 10) are Pythagorean triples.

Here Katya wondered if she can specify the length of some side of right triangle and find any Pythagorean triple corresponding to such length? Note that the side which length is specified can be a cathetus as well as hypotenuse.

Katya had no problems with completing this task. Will you do the same?

**Input**

The only line of the input contains single integer *n* (1 ≤ *n* ≤ 109) — the length of some side of a right triangle.

**Output**

Print two integers *m* and *k* (1 ≤ *m*, *k* ≤ 1018), such that *n*, *m* and *k* form a Pythagorean triple, in the only line.

In case if there is no any Pythagorean triple containing integer *n*, print  - 1 in the only line. If there are many answers, print any of them.

**Examples**

**input**

3

**output**

4 5

**input**

6

**output**

8 10

**input**

1

**output**

-1

**input**

17

**output**

144 145

**input**

67

**output**

2244 2245

**Note**

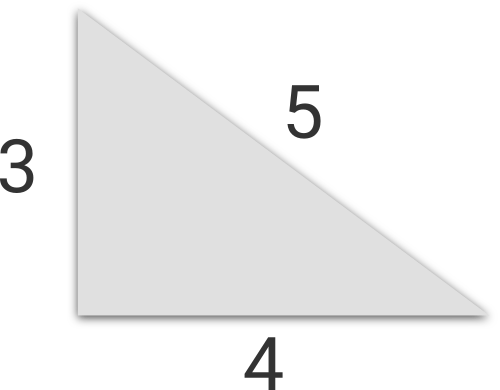


Illustration for the first sample.

B. Bakery

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Masha wants to open her own bakery and bake muffins in one of the *n* cities numbered from 1 to *n*. There are *m* bidirectional roads, each of whose connects some pair of cities.

To bake muffins in her bakery, Masha needs to establish flour supply from some storage. There are only *k* storages, located in different cities numbered *a*1, *a*2, ..., *ak*.

Unforunately the law of the country Masha lives in prohibits opening bakery in any of the cities which has storage located in it. She can open it only in one of another *n* - *k* cities, and, of course, flour delivery should be paid — for every kilometer of path between storage and bakery Masha should pay 1 ruble.

Formally, Masha will pay *x* roubles, if she will open the bakery in some city *b* (*ai* ≠ *b* for every 1 ≤ *i* ≤ *k*) and choose a storage in some city *s* (*s* = *aj* for some 1 ≤ *j* ≤ *k*) and *b* and *s* are connected by some path of roads of summary length *x* (if there are more than one path, Masha is able to choose which of them should be used).

Masha is very thrifty and rational. She is interested in a city, where she can open her bakery (and choose one of *k* storages and one of the paths between city with bakery and city with storage) and pay minimum possible amount of rubles for flour delivery. Please help Masha find this amount.

**Input**

The first line of the input contains three integers *n*, *m* and *k* (1 ≤ *n*, *m* ≤ 105, 0 ≤ *k* ≤ *n*) — the number of cities in country Masha lives in, the number of roads between them and the number of flour storages respectively.

Then *m* lines follow. Each of them contains three integers *u*, *v* and *l* (1 ≤ *u*, *v* ≤ *n*, 1 ≤ *l* ≤ 109, *u* ≠ *v*) meaning that there is a road between cities *u* and *v* of length of *l* kilometers .

If *k* > 0, then the last line of the input contains *k* distinct integers *a*1, *a*2, ..., *ak* (1 ≤ *ai* ≤ *n*) — the number of cities having flour storage located in. If *k* = 0 then this line **is not presented in the input**.

**Output**

Print the minimum possible amount of rubles Masha should pay for flour delivery in the only line.

If the bakery can not be opened (while satisfying conditions) in any of the *n* cities, print  - 1 in the only line.

**Examples**

**input**

5 4 2  
1 2 5  
1 2 3  
2 3 4  
1 4 10  
1 5

**output**

3

**input**

3 1 1  
1 2 3  
3

**output**

-1

**Note**

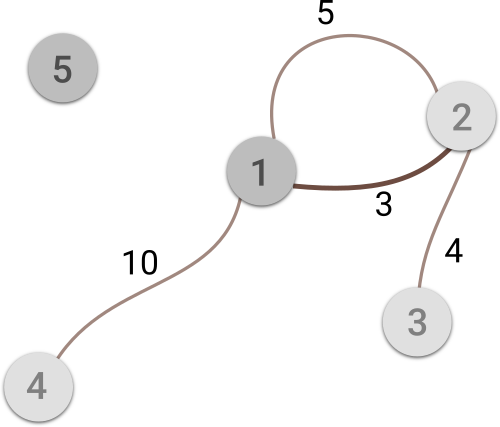


Image illustrates the first sample case. Cities with storage located in and the road representing the answer are darkened.

A. Brain's Photos

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Small, but very brave, mouse Brain was not accepted to summer school of young villains. He was upset and decided to postpone his plans of taking over the world, but to become a photographer instead.

As you may know, the coolest photos are on the film (because you can specify the hashtag #film for such).

Brain took a lot of colourful pictures on colored and black-and-white film. Then he developed and translated it into a digital form. But now, color and black-and-white photos are in one folder, and to sort them, one needs to spend more than one hour!

As soon as Brain is a photographer not programmer now, he asks you to help him determine for a **single** photo whether it is colored or black-and-white.

Photo can be represented as a matrix sized *n* × *m*, and each element of the matrix stores a symbol indicating corresponding pixel color. There are only 6 colors:

* 'C' (cyan)
* 'M' (magenta)
* 'Y' (yellow)
* 'W' (white)
* 'G' (grey)
* 'B' (black)

The photo is considered black-and-white if it has only white, black and grey pixels in it. If there are any of cyan, magenta or yellow pixels in the photo then it is considered colored.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 100) — the number of photo pixel matrix rows and columns respectively.

Then *n* lines describing matrix rows follow. Each of them contains *m* space-separated characters describing colors of pixels in a row. Each character in the line is one of the 'C', 'M', 'Y', 'W', 'G' or 'B'.

**Output**

Print the "#Black&White" (without quotes), if the photo is black-and-white and "#Color" (without quotes), if it is colored, in the only line.

**Examples**

**input**

2 2  
C M  
Y Y

**output**

#Color

**input**

3 2  
W W  
W W  
B B

**output**

#Black&White

**input**

1 1  
W

**output**

#Black&White

E. Working routine

time limit per test

2.5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasiliy finally got to work, where there is a huge amount of tasks waiting for him. Vasiliy is given a matrix consisting of *n* rows and *m*columns and *q* tasks. Each task is to swap two submatrices of the given matrix.

For each task Vasiliy knows six integers *ai*, *bi*, *ci*, *di*, *hi*, *wi*, where *ai* is the index of the row where the top-left corner of the first rectangle is located, *bi* is the index of its column, *ci* is the index of the row of the top-left corner of the second rectangle, *di* is the index of its column, *hi* is the height of the rectangle and *wi* is its width.

It's guaranteed that two rectangles in one query do not overlap and do not touch, that is, no cell belongs to both rectangles, and no two cells belonging to different rectangles **share a side**. However, rectangles are allowed to share an angle.

Vasiliy wants to know how the matrix will look like after all tasks are performed.

**Input**

The first line of the input contains three integers *n*, *m* and *q* (2 ≤ *n*, *m* ≤ 1000, 1 ≤ *q* ≤ 10 000) — the number of rows and columns in matrix, and the number of tasks Vasiliy has to perform.

Then follow *n* lines containing *m* integers *vi*,*j* (1 ≤ *vi*,*j* ≤ 109) each — initial values of the cells of the matrix.

Each of the following *q* lines contains six integers *ai*, *bi*, *ci*, *di*, *hi*, *wi* (1 ≤ *ai*, *ci*, *hi* ≤ *n*, 1 ≤ *bi*, *di*, *wi* ≤ *m*).

**Output**

Print *n* lines containing *m* integers each — the resulting matrix.

**Examples**

**input**

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

**output**

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

**input**

4 2 1  
1 1  
1 1  
2 2  
2 2  
1 1 4 1 1 2

**output**

2 2  
1 1  
2 2  
1 1

D. Vasiliy's Multiset

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Author has gone out of the stories about Vasiliy, so here is just a formal task description.

You are given *q* queries and a multiset *A*, initially containing only integer 0. There are three types of queries:

1. "+ x" — add integer *x* to multiset *A*.
2. "- x" — erase one occurrence of integer *x* from multiset *A*. It's guaranteed that at least one *x* is present in the multiset *A* before this query.
3. "? x" — you are given integer *x* and need to compute the value http://codeforces.com/predownloaded/af/1b/af1b4d5401ef843e35de22bbd1367fa78816de31.png, i.e. the maximum value of bitwise exclusive OR (also know as XOR) of integer *x* and some integer *y* from the multiset *A*.

Multiset is a set, where equal elements are allowed.

**Input**

The first line of the input contains a single integer *q* (1 ≤ *q* ≤ 200 000) — the number of queries Vasiliy has to perform.

Each of the following *q* lines of the input contains one of three characters '+', '-' or '?' and an integer *xi* (1 ≤ *xi* ≤ 109). It's guaranteed that there is at least one query of the third type.

Note, that the integer 0 will always be present in the set *A*.

**Output**

For each query of the type '?' print one integer — the maximum value of bitwise exclusive OR (XOR) of integer *xi* and some integer from the multiset *A*.

**Example**

**input**

10  
+ 8  
+ 9  
+ 11  
+ 6  
+ 1  
? 3  
- 8  
? 3  
? 8  
? 11

**output**

11  
10  
14  
13

**Note**

After first five operations multiset *A* contains integers 0, 8, 9, 11, 6 and 1.

The answer for the sixth query is integer http://codeforces.com/predownloaded/f1/be/f1be3f0eb002b70f2885010a36a4ee3437035f76.png — maximum among integers http://codeforces.com/predownloaded/37/b9/37b978f8f58974079cd14c236f215799c4498de1.png, http://codeforces.com/predownloaded/5d/c1/5dc10ff70d05fbc23f9ca460fcdaabd323da2000.png, http://codeforces.com/predownloaded/25/ea/25eadc76e65c2dcc9e605d7cbcb8b85c432154b6.png, http://codeforces.com/predownloaded/4e/9f/4e9f7b2b7c9c3c4a682f5417c5422483016ee463.png and http://codeforces.com/predownloaded/6f/0d/6f0d4181cb4df733db930b2d79c66c1f83fafd42.png.

C. Hard problem

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasiliy is fond of solving different tasks. Today he found one he wasn't able to solve himself, so he asks you to help.

Vasiliy is given *n* strings consisting of lowercase English letters. He wants them to be sorted in lexicographical order (as in the dictionary), but he is not allowed to swap any of them. The only operation he is allowed to do is to reverse any of them (first character becomes last, second becomes one before last and so on).

To reverse the *i*-th string Vasiliy has to spent *ci* units of energy. He is interested in the minimum amount of energy he has to spent in order to have strings sorted in lexicographical order.

String *A* is lexicographically smaller than string *B* if it is shorter than *B* (|*A*| < |*B*|) and is its prefix, or if none of them is a prefix of the other and at the first position where they differ character in *A* is smaller than the character in *B*.

For the purpose of this problem, two equal strings nearby do not break the condition of sequence being sorted lexicographically.

**Input**

The first line of the input contains a single integer *n* (2 ≤ *n* ≤ 100 000) — the number of strings.

The second line contains *n* integers *ci* (0 ≤ *ci* ≤ 109), the *i*-th of them is equal to the amount of energy Vasiliy has to spent in order to reverse the *i*-th string.

Then follow *n* lines, each containing a string consisting of lowercase English letters. The total length of these strings doesn't exceed 100 000.

**Output**

If it is impossible to reverse some of the strings such that they will be located in lexicographical order, print  - 1. Otherwise, print the minimum total amount of energy Vasiliy has to spent.

**Examples**

**input**

2  
1 2  
ba  
ac

**output**

1

**input**

3  
1 3 1  
aa  
ba  
ac

**output**

1

**input**

2  
5 5  
bbb  
aaa

**output**

-1

**input**

2  
3 3  
aaa  
aa

**output**

-1

**Note**

In the second sample one has to reverse string 2 or string 3. To amount of energy required to reverse the string 3 is smaller.

In the third sample, both strings do not change after reverse and they go in the wrong order, so the answer is  - 1.

In the fourth sample, both strings consists of characters 'a' only, but in the sorted order string "aa" should go before string "aaa", thus the answer is  - 1.

B. Interesting drink

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasiliy likes to rest after a hard work, so you may often meet him in some bar nearby. As all programmers do, he loves the famous drink "Beecola", which can be bought in *n* different shops in the city. It's known that the price of one bottle in the shop *i* is equal to *xi* coins.

Vasiliy plans to buy his favorite drink for *q* consecutive days. He knows, that on the *i*-th day he will be able to spent *mi* coins. Now, for each of the days he want to know in how many different shops he can buy a bottle of "Beecola".

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the number of shops in the city that sell Vasiliy's favourite drink.

The second line contains *n* integers *xi* (1 ≤ *xi* ≤ 100 000) — prices of the bottles of the drink in the *i*-th shop.

The third line contains a single integer *q* (1 ≤ *q* ≤ 100 000) — the number of days Vasiliy plans to buy the drink.

Then follow *q* lines each containing one integer *mi* (1 ≤ *mi* ≤ 109) — the number of coins Vasiliy can spent on the *i*-th day.

**Output**

Print *q* integers. The *i*-th of them should be equal to the number of shops where Vasiliy will be able to buy a bottle of the drink on the *i*-th day.

**Example**

**input**

5  
3 10 8 6 11  
4  
1  
10  
3  
11

**output**

0  
4  
1  
5

**Note**

On the first day, Vasiliy won't be able to buy a drink in any of the shops.

On the second day, Vasiliy can buy a drink in the shops 1, 2, 3 and 4.

On the third day, Vasiliy can buy a drink only in the shop number 1.

Finally, on the last day Vasiliy can buy a drink in any shop.

A. Beru-taxi

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasiliy lives at point (*a*, *b*) of the coordinate plane. He is hurrying up to work so he wants to get out of his house as soon as possible. New app suggested *n* available Beru-taxi nearby. The *i*-th taxi is located at point (*xi*, *yi*) and moves with a speed *vi*.

Consider that each of *n* drivers will move directly to Vasiliy and with a maximum possible speed. Compute the minimum time when Vasiliy will get in any of Beru-taxi cars.

**Input**

The first line of the input contains two integers *a* and *b* ( - 100 ≤ *a*, *b* ≤ 100) — coordinates of Vasiliy's home.

The second line contains a single integer *n* (1 ≤ *n* ≤ 1000) — the number of available Beru-taxi cars nearby.

The *i*-th of the following *n* lines contains three integers *xi*, *yi* and *vi* ( - 100 ≤ *xi*, *yi* ≤ 100, 1 ≤ *vi* ≤ 100) — the coordinates of the *i*-th car and its speed.

It's allowed that several cars are located at the same point. Also, cars may be located at exactly the same point where Vasiliy lives.

**Output**

Print a single real value — the minimum time Vasiliy needs to get in any of the Beru-taxi cars. You answer will be considered correct if its absolute or relative error does not exceed 10- 6.

Namely: let's assume that your answer is *a*, and the answer of the jury is *b*. The checker program will consider your answer correct, if http://codeforces.com/predownloaded/c6/2e/c62ea64d4651240724c5ac4779b671c741edec24.png.

**Examples**

**input**

0 0  
2  
2 0 1  
0 2 2

**output**

1.00000000000000000000

**input**

1 3  
3  
3 3 2  
-2 3 6  
-2 7 10

**output**

0.50000000000000000000

**Note**

In the first sample, first taxi will get to Vasiliy in time 2, and second will do this in time 1, therefore 1 is the answer.

In the second sample, cars 2 and 3 will arrive simultaneously.

B. Spider Man

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Peter Parker wants to play a game with Dr. Octopus. The game is about cycles. *Cycle* is a sequence of vertices, such that first one is connected with the second, second is connected with third and so on, while the last one is connected with the first one again. Cycle may consist of a single isolated vertex.

Initially there are *k* cycles, *i*-th of them consisting of exactly *vi* vertices. Players play alternatively. Peter goes first. On each turn a player must choose a cycle with at least 2 vertices (for example, *x* vertices) among all available cycles and replace it by two cycles with *p* and *x* - *p* vertices where 1 ≤ *p* < *x* is chosen by the player. The player who cannot make a move loses the game (and his life!).

Peter wants to test some configurations of initial cycle sets before he actually plays with Dr. Octopus. Initially he has an empty set. In the *i*-th test he adds a cycle with *ai* vertices to the set (this is actually a multiset because it can contain two or more identical cycles). After each test, Peter wants to know that if the players begin the game with the current set of cycles, who wins?

Peter is pretty good at math, but now he asks you to help.

**Input**

The first line of the input contains a single integer *n* (1 ≤ *n* ≤ 100 000) — the number of tests Peter is about to make.

The second line contains *n* space separated integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109), *i*-th of them stands for the number of vertices in the cycle added before the *i*-th test.

**Output**

Print the result of all tests in order they are performed. Print 1 if the player who moves first wins or 2 otherwise.

**Examples**

**input**

3  
1 2 3

**output**

2  
1  
1

**input**

5  
1 1 5 1 1

**output**

2  
2  
2  
2  
2

**Note**

In the first sample test:

In Peter's first test, there's only one cycle with 1 vertex. First player cannot make a move and loses.

In his second test, there's one cycle with 1 vertex and one with 2. No one can make a move on the cycle with 1 vertex. First player can replace the second cycle with two cycles of 1 vertex and second player can't make any move and loses.

In his third test, cycles have 1, 2 and 3 vertices. Like last test, no one can make a move on the first cycle. First player can replace the third cycle with one cycle with size 1 and one with size 2. Now cycles have 1, 1, 2, 2 vertices. Second player's only move is to replace a cycle of size 2 with 2 cycles of size 1. And cycles are 1, 1, 1, 1, 2. First player replaces the last cycle with 2 cycles with size 1 and wins.

In the second sample test:

Having cycles of size 1 is like not having them (because no one can make a move on them).

In Peter's third test: There a cycle of size 5 (others don't matter). First player has two options: replace it with cycles of sizes 1 and 4 or 2and 3.

* If he replaces it with cycles of sizes 1 and 4: Only second cycle matters. Second player will replace it with 2 cycles of sizes 2. First player's only option to replace one of them with two cycles of size 1. Second player does the same thing with the other cycle. First player can't make any move and loses.
* If he replaces it with cycles of sizes 2 and 3: Second player will replace the cycle of size 3 with two of sizes 1 and 2. Now only cycles with more than one vertex are two cycles of size 2. As shown in previous case, with 2 cycles of size 2 second player wins.

So, either way first player loses.

A. Hulk

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Dr. Bruce Banner hates his enemies (like others don't). As we all know, he can barely talk when he turns into the incredible Hulk. That's why he asked you to help him to express his feelings.

Hulk likes the Inception so much, and like that his feelings are complicated. They have *n* layers. The first layer is hate, second one is love, third one is hate and so on...

For example if *n* = 1, then his feeling is "I hate it" or if *n* = 2 it's "I hate that I love it", and if *n* = 3 it's "I hate that I love that I hate it" and so on.

Please help Dr. Banner.

**Input**

The only line of the input contains a single integer *n* (1 ≤ *n* ≤ 100) — the number of layers of love and hate.

**Output**

Print Dr.Banner's feeling in one line.

**Examples**

**input**

1

**output**

I hate it

**input**

2

**output**

I hate that I love it

**input**

3

**output**

I hate that I love that I hate it

E. Iron Man

time limit per test

5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Tony Stark is playing a game with his suits (they have auto-pilot now). He lives in Malibu. Malibu has *n* junctions numbered from 1 to *n*, connected with *n* - 1 roads. One can get from a junction to any other junction using these roads (graph of Malibu forms a tree).

Tony has *m* suits. There's a special plan for each suit. The *i*-th suit will appear at the moment of time *ti* in the junction *vi*, and will move to junction *ui* using the shortest path between *vi* and *ui* with the speed *ci* roads per second (passing a junctions takes no time), and vanishing immediately when arriving at *ui* (if it reaches *ui* in time *q*, it's available there at moment *q*, but not in further moments). Also, suits move continuously (for example if *vi* ≠ *ui*, at time http://codeforces.com/predownloaded/e7/1f/e71ff9cf57a2d8eb419b65095d7700e94133f384.png it's in the middle of a road. Please note that if *vi* = *ui* it means the suit will be at junction number *vi* only at moment *ti* and then it vanishes.

An explosion happens if at any moment of time two suits share the same exact location (it may be in a junction or somewhere on a road; while appearing, vanishing or moving).

Your task is to tell Tony the moment of the the first explosion (if there will be any).

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 100 000) — the number of junctions and the number of suits respectively.

The next *n* - 1 lines contain the roads descriptions. Each line contains two integers *ai* and *bi* — endpoints of the *i*-th road (1 ≤ *ai*, *bi* ≤ *n*, *ai* ≠ *bi*).

The next *m* lines contain the suit descriptions. The *i*-th of them contains four integers *ti*, *ci*, *vi* and *ui* (0 ≤ *ti* ≤ 10 000, 1 ≤ *ci* ≤ 10 000, 1 ≤ *vi*, *ui* ≤ *n*), meaning the *i*-th suit will appear at moment of time *ti* at the junction *vi* and will move to the junction *ui* with a speed *ci*roads per second.

**Output**

If there would be no explosions at all, print -1 in the first and only line of output.

Otherwise print the moment of the first explosion.

Your answer will be considered correct if its relative or absolute error doesn't exceed 10- 6.

**Examples**

**input**

6 4  
2 5  
6 5  
3 6  
4 6  
4 1  
27 6 1 3  
9 5 1 6  
27 4 3 4  
11 29 2 6

**output**

27.3

**input**

6 4  
3 1  
4 5  
6 4  
6 1  
2 6  
16 4 4 5  
13 20 6 2  
3 16 4 5  
28 5 3 5

**output**

-1

D. Captain America

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Steve Rogers is fascinated with new vibranium shields S.H.I.E.L.D gave him. They're all uncolored. There are *n* shields in total, the *i*-th shield is located at point (*xi*, *yi*) of the coordinate plane. It's possible that two or more shields share the same location.

Steve wants to paint all these shields. He paints each shield in either red or blue. Painting a shield in red costs *r* dollars while painting it in blue costs *b* dollars.

Additionally, there are *m* constraints Steve wants to be satisfied. The *i*-th constraint is provided by three integers *ti*, *li* and *di*:

* If *ti* = 1, then the absolute difference between the number of red and blue shields on line *x* = *li* should not exceed *di*.
* If *ti* = 2, then the absolute difference between the number of red and blue shields on line *y* = *li* should not exceed *di*.

Steve gave you the task of finding the painting that satisfies all the condition and the total cost is minimum.

**Input**

The first line of the input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 100 000) — the number of shields and the number of constraints respectively.

The second line contains two integers *r* and *b* (1 ≤ *r*, *b* ≤ 109).

The next *n* lines contain the shields coordinates. The *i*-th of these lines contains two integers *xi* and *yi* (1 ≤ *xi*, *yi* ≤ 109).

The next *m* lines contain the constrains. The *j*-th of these lines contains three integers *tj*, *lj* and *dj* (1 ≤ *tj* ≤ 2, 1 ≤ *lj* ≤ 109, 0 ≤ *dj* ≤ *n*).

**Output**

If satisfying all the constraints is impossible print -1 in first and only line of the output.

Otherwise, print the minimum total cost in the first line of output. In the second line print a string of length *n* consisting of letters 'r' and 'b' only. The *i*-th character should be 'r' if the *i*-th shield should be painted red in the optimal answer and 'b' if it should be painted blue. The cost of painting shields in these colors should be equal the minimum cost you printed on the first line.

If there exist more than one optimal solution, print any of them.

**Examples**

**input**

5 6  
8 3  
2 10  
1 5  
9 10  
9 10  
2 8  
1 9 1  
1 2 1  
2 10 3  
2 10 2  
1 1 1  
2 5 2

**output**

25  
rbrbb

**input**

4 4  
7 3  
10 3  
9 8  
10 3  
2 8  
2 8 0  
2 8 0  
1 2 0  
1 9 0

**output**

-1

C. Black Widow

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Natalia Romanova is trying to test something on the new gun S.H.I.E.L.D gave her. In order to determine the result of the test, she needs to find the number of answers to a certain equation. The equation is of form:

http://codeforces.com/predownloaded/0d/63/0d637ea7cd9d50e60ff47db836c3a91e0eb849b0.png

Where http://codeforces.com/predownloaded/f6/6d/f66d5a8baeb3645356745f84de710ee8bca95c8d.png represents logical OR and http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png represents logical exclusive OR (XOR), and *vi*,*j* are some boolean variables or their negations. Natalia calls the left side of the equation a XNF formula. Each statement in brackets is called a clause, and *vi*,*j* are called literals.

In the equation Natalia has, the left side is actually a 2-XNF-2 containing variables *x*1, *x*2, ..., *xm* and their negations. An XNF formula is 2-XNF-2 if:

1. For each 1 ≤ *i* ≤ *n*, *ki* ≤ 2, i.e. the size of each clause doesn't exceed two.
2. Each variable occurs **in the formula at most two times** (with negation and without negation in total). Please note that it's possible that a variable occurs twice but its negation doesn't occur in any clause (or vice versa).

Natalia is given a formula of *m* variables, consisting of *n* clauses. Please, make sure to check the samples in order to properly understand how the formula looks like.

Natalia is more into fight than theory, so she asked you to tell her the number of answers to this equation. More precisely, you need to find the number of ways to set *x*1, ..., *xm* with *true* and *false* (out of total of 2*m* ways) so that the equation is satisfied. Since this number can be extremely large, you need to print the answer modulo 109 + 7.

Please, note that some variable may appear twice in one clause, or not appear in the equation at all (but still, setting it to *false* or *true*gives different ways to set variables).

**Input**

The first line of input contains two integers *n* and *m* (1 ≤ *n*, *m* ≤ 100 000) — the number of clauses and the number of variables respectively.

The next *n* lines contain the formula. The *i*-th of them starts with an integer *ki* — the number of literals in the *i*-th clause. It is followed by *ki* non-zero integers *ai*, 1, ..., *ai*,*ki*. If *ai*,*j* > 0 then *vi*,*j* is *xai*,*j* otherwise it's negation of *x*-*ai*,*j* (1 ≤ *ki* ≤ 2,  - *m* ≤ *ai*,*j* ≤ *m*, *ai*,*j* ≠ 0).

**Output**

Print the answer modulo 1 000 000 007 (109 + 7) in one line.

**Examples**

**input**

6 7  
2 4 -2  
2 6 3  
2 -7 1  
2 -5 1  
2 3 6  
2 -2 -5

**output**

48

**input**

8 10  
1 -5  
2 4 -6  
2 -2 -6  
2 -7 9  
2 10 -1  
2 3 -1  
2 -8 9  
2 5 8

**output**

544

**input**

2 3  
2 1 1  
2 -3 3

**output**

4

**Note**

The equation in the first sample is:

http://codeforces.com/predownloaded/28/8f/288f07e976c1475cdda679f67942ae7779907997.png

The equation in the second sample is:

http://codeforces.com/predownloaded/16/97/1697cc609520b34073afa6e868af4687f02d96a5.png

The equation in the third sample is:

http://codeforces.com/predownloaded/14/91/149195216145c6b69c5cf61dd873530f481f8a68.png

B. Ant Man

704B

time limit per test

4 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Scott Lang is at war with Darren Cross. There are *n* chairs in a hall where they are, numbered with 1, 2, ..., *n* from left to right. The *i*-th chair is located at coordinate *xi*. Scott is on chair number *s* and Cross is on chair number *e*. Scott can jump to all other chairs (not only neighboring chairs). He wants to start at his position (chair number *s*), visit each chair **exactly once** and end up on chair number *e* with Cross.

As we all know, Scott can shrink or grow big (grow big only to his normal size), so at any moment of time he can be either small or large (normal). The thing is, he can only shrink or grow big while being on a chair (not in the air while jumping to another chair). Jumping takes time, but shrinking and growing big takes no time. Jumping from chair number *i* to chair number *j* takes |*xi* - *xj*| seconds. Also, jumping off a chair and landing on a chair takes extra amount of time.

If Scott wants to jump to a chair on his left, he can only be small, and if he wants to jump to a chair on his right he should be large.

Jumping off the *i*-th chair takes:

* *ci* extra seconds if he's small.
* *di* extra seconds otherwise (he's large).

Also, landing on *i*-th chair takes:

* *bi* extra seconds if he's small.
* *ai* extra seconds otherwise (he's large).

In simpler words, jumping from *i*-th chair to *j*-th chair takes exactly:

* |*xi* - *xj*| + *ci* + *bj* seconds if *j* < *i*.
* |*xi* - *xj*| + *di* + *aj* seconds otherwise (*j* > *i*).

Given values of *x*, *a*, *b*, *c*, *d* find the minimum time Scott can get to Cross, assuming he wants to visit each chair exactly once.

**Input**

The first line of the input contains three integers *n*, *s* and *e* (2 ≤ *n* ≤ 5000, 1 ≤ *s*, *e* ≤ *n*, *s* ≠ *e*) — the total number of chairs, starting and ending positions of Scott.

The second line contains *n* integers *x*1, *x*2, ..., *xn* (1 ≤ *x*1 < *x*2 < ... < *xn* ≤ 109).

The third line contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *a*1, *a*2, ..., *an* ≤ 109).

The fourth line contains *n* integers *b*1, *b*2, ..., *bn* (1 ≤ *b*1, *b*2, ..., *bn* ≤ 109).

The fifth line contains *n* integers *c*1, *c*2, ..., *cn* (1 ≤ *c*1, *c*2, ..., *cn* ≤ 109).

The sixth line contains *n* integers *d*1, *d*2, ..., *dn* (1 ≤ *d*1, *d*2, ..., *dn* ≤ 109).

**Output**

Print the minimum amount of time Scott needs to get to the Cross while visiting each chair exactly once.

**Example**

**input**

7 4 3  
8 11 12 16 17 18 20  
17 16 20 2 20 5 13  
17 8 8 16 12 15 13  
12 4 16 4 15 7 6  
8 14 2 11 17 12 8

**output**

139

**Note**

In the sample testcase, an optimal solution would be http://codeforces.com/predownloaded/9a/2c/9a2c9cc57ddd19f3cc63a835f9a9a7ddf11d438f.png. Spent time would be 17 + 24 + 23 + 20 + 33 + 22 = 139.

A. Thor

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Thor is getting used to the Earth. As a gift Loki gave him a smartphone. There are *n* applications on this phone. Thor is fascinated by this phone. He has only one minor issue: he can't count the number of unread notifications generated by those applications (maybe Loki put a curse on it so he can't).

*q* events are about to happen (in chronological order). They are of three types:

1. Application *x* generates a notification (this new notification is unread).
2. Thor reads all notifications generated so far by application *x* (he may re-read some notifications).
3. Thor reads the first *t* notifications generated by phone applications (notifications generated in first *t* events of the first type). It's guaranteed that there were at least *t* events of the first type before this event. Please note that he doesn't read first *t* unread notifications, he just reads the very first *t* notifications generated on his phone and he may re-read some of them in this operation.

Please help Thor and tell him the number of unread notifications after each event. You may assume that initially there are no notifications in the phone.

**Input**

The first line of input contains two integers *n* and *q* (1 ≤ *n*, *q* ≤ 300 000) — the number of applications and the number of events to happen.

The next *q* lines contain the events. The *i*-th of these lines starts with an integer *typei* — type of the *i*-th event. If *typei* = 1 or *typei* = 2then it is followed by an integer *xi*. Otherwise it is followed by an integer *ti* (1 ≤ *typei* ≤ 3, 1 ≤ *xi* ≤ *n*, 1 ≤ *ti* ≤ *q*).

**Output**

Print the number of unread notifications after each event.

**Examples**

**input**

3 4  
1 3  
1 1  
1 2  
2 3

**output**

1  
2  
3  
2

**input**

4 6  
1 2  
1 4  
1 2  
3 3  
1 3  
1 3

**output**

1  
2  
3  
0  
1  
2

**Note**

In the first sample:

1. Application 3 generates a notification (there is 1 unread notification).
2. Application 1 generates a notification (there are 2 unread notifications).
3. Application 2 generates a notification (there are 3 unread notifications).
4. Thor reads the notification generated by application 3, there are 2 unread notifications left.

In the second sample test:

1. Application 2 generates a notification (there is 1 unread notification).
2. Application 4 generates a notification (there are 2 unread notifications).
3. Application 2 generates a notification (there are 3 unread notifications).
4. Thor reads first three notifications and since there are only three of them so far, there will be no unread notification left.
5. Application 3 generates a notification (there is 1 unread notification).
6. Application 3 generates a notification (there are 2 unread notifications).

E. Mishka and Divisors

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

After playing with her beautiful array, Mishka decided to learn some math. After learning how to multiply, divide and what is divisibility, she is now interested in solving the following problem.

You are given integer *k* and array *a*1, *a*2, ..., *an* of *n* integers. You are to find **non-empty** subsequence of array elements such that the product of its elements is divisible by *k* and it contains minimum possible number of elements.

Formally, you are to find a sequence of indices 1 ≤ *i*1 < *i*2 < ... < *im* ≤ *n* such that http://codeforces.com/predownloaded/2b/ec/2bec6578cb577365a1915ce3f256f59319071cce.png is divisible by *k* while *m* is minimum possible among all such variants.

If there are more than one such subsequences, you should choose one among them, such that sum of its elements is **minimum possible**.

Mishka quickly solved this problem. Will you do so?

**Input**

The first line of the input contains two integers *n* and *k* (1 ≤ *n* ≤ 1 000, 1 ≤ *k* ≤ 1012).

The second line of the input contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 1012) — array elements.

**Output**

Print single positive integer *m* in the first line — the number of elements in desired sequence.

In the second line print *m* distinct integers — the sequence of indices of given array elements, which should be taken into the desired sequence.

If there are more than one such subsequence (e.g. subsequence of minimum possible number of elements and with minimum possible sum of elements), you can print any of them.

If there are no such subsequences, print  - 1 in the only line.

**Example**

**input**

5 60  
2 4 6 5 2

**output**

3  
4 3 1

D. Mishka and Interesting sum

time limit per test

3.5 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Little Mishka enjoys programming. Since her birthday has just passed, her friends decided to present her with array of non-negative integers *a*1, *a*2, ..., *an* of *n* elements!

Mishka loved the array and she instantly decided to determine its beauty value, but she is too little and can't process large arrays. Right because of that she invited you to visit her and asked you to process *m* queries.

Each query is processed in the following way:

1. Two integers *l* and *r* (1 ≤ *l* ≤ *r* ≤ *n*) are specified — bounds of query segment.
2. Integers, presented in array segment [*l*,  *r*] (in sequence of integers *al*, *al*+ 1, ..., *ar*) **even number of times**, are written down.
3. XOR-sum of written down integers is calculated, and this value is the answer for a query. Formally, if integers written down in point 2 are *x*1, *x*2, ..., *xk*, then Mishka wants to know the value http://codeforces.com/predownloaded/ea/45/ea4547ffed8a3c8eb751acf0531e94388122e790.png, where http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png — operator of exclusive bitwise OR.

Since only the little bears know the definition of array beauty, all you are to do is to answer each of queries presented.

**Input**

The first line of the input contains single integer *n* (1 ≤ *n* ≤ 1 000 000) — the number of elements in the array.

The second line of the input contains *n* integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109) — array elements.

The third line of the input contains single integer *m* (1 ≤ *m* ≤ 1 000 000) — the number of queries.

Each of the next *m* lines describes corresponding query by a pair of integers *l* and *r* (1 ≤ *l* ≤ *r* ≤ *n*) — the bounds of query segment.

**Output**

Print *m* non-negative integers — the answers for the queries in the order they appear in the input.

**Examples**

**input**

3  
3 7 8  
1  
1 3

**output**

0

**input**

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

**output**

0  
3  
1  
3  
2

**Note**

In the second sample:

There is no integers in the segment of the first query, presented even number of times in the segment — the answer is 0.

In the second query there is only integer 3 is presented even number of times — the answer is 3.

In the third query only integer 1 is written down — the answer is 1.

In the fourth query all array elements are considered. Only 1 and 2 are presented there even number of times. The answer is http://codeforces.com/predownloaded/f3/f4/f3f48ca9259e359e0f860214756305ac8291888b.png.

In the fifth query 1 and 3 are written down. The answer is http://codeforces.com/predownloaded/a1/ba/a1ba125fbec1b545761705318974b0be213f2ef2.png.

C. Chris and Road

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

And while Mishka is enjoying her trip...

Chris is a little brown bear. No one knows, where and when he met Mishka, but for a long time they are together (excluding her current trip). However, best friends are important too. John is Chris' best friend.

Once walking with his friend, John gave Chris the following problem:

At the infinite horizontal road of width *w*, bounded by lines *y* = 0 and *y* = *w*, there is a bus moving, presented as a convex polygon of *n*vertices. The bus moves continuously with a constant speed of *v* in a straight *Ox* line in direction of decreasing *x* coordinates, thus in time **only *x* coordinates** of its points are changing. Formally, after time *t* each of *x* coordinates of its points will be decreased by *vt*.

There is a pedestrian in the point (0, 0), who can move only by a vertical pedestrian crossing, presented as a segment connecting points (0, 0) and (0, *w*) with any speed not exceeding *u*. Thus the pedestrian can move only in a straight line *Oy* in any direction with any speed not exceeding *u* and not leaving the road borders. The pedestrian can instantly change his speed, thus, for example, he can stop instantly.

Please look at the sample note picture for better understanding.

We consider the pedestrian is *hit by the bus*, if at any moment the point he is located in lies **strictly inside** the bus polygon (this means that if the point lies on the polygon vertex or on its edge, the pedestrian is not hit by the bus).

You are given the bus position at the moment 0. Please help Chris determine minimum amount of time the pedestrian needs to cross the road and reach the point (0, *w*) and not to be hit by the bus.

**Input**

The first line of the input contains four integers *n*, *w*, *v*, *u* (3 ≤ *n* ≤ 10 000, 1 ≤ *w* ≤ 109, 1 ≤ *v*,  *u* ≤ 1000) — the number of the bus polygon vertices, road width, bus speed and pedestrian speed respectively.

The next *n* lines describes polygon vertices in counter-clockwise order. *i*-th of them contains pair of integers *xi* and *yi* ( - 109 ≤ *xi* ≤ 109, 0 ≤ *yi* ≤ *w*) — coordinates of *i*-th polygon point. It is guaranteed that the polygon is non-degenerate.

**Output**

Print the single real *t* — the time the pedestrian needs to croos the road and not to be hit by the bus. The answer is considered correct if its relative or absolute error doesn't exceed 10- 6.

**Example**

**input**

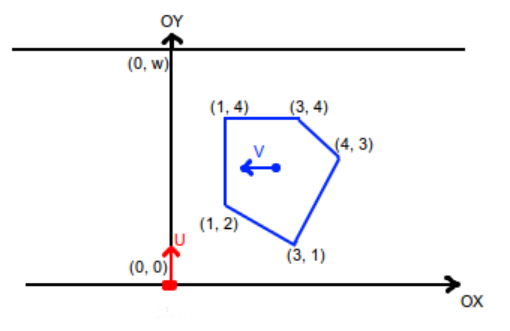
5 5 1 2  
1 2  
3 1  
4 3  
3 4  
1 4

**output**

5.0000000000

**Note**

Following image describes initial position in the first sample case:



B. Mishka and trip

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Little Mishka is a great traveller and she visited many countries. After thinking about where to travel this time, she chose XXX — beautiful, but little-known northern country.

Here are some interesting facts about XXX:

1. XXX consists of *n* cities, *k* of whose (just imagine!) are capital cities.
2. All of cities in the country are beautiful, but each is beautiful in its own way. Beauty value of *i*-th city equals to *ci*.
3. All the cities are consecutively connected by the roads, including 1-st and *n*-th city, forming a cyclic route 1 — 2 — ... — *n* — 1. Formally, for every 1 ≤ *i* < *n* there is a road between *i*-th and *i* + 1-th city, and another one between 1-st and *n*-th city.
4. Each capital city is connected with each other city directly by the roads. Formally, if city *x* is a capital city, then for every 1 ≤ *i* ≤ *n*,  *i* ≠ *x*, there is a road between cities *x* and *i*.
5. There is **at most one** road between any two cities.
6. Price of passing a road directly depends on beauty values of cities it connects. Thus if there is a road between cities *i* and *j*, price of passing it equals *ci*·*cj*.

Mishka started to gather her things for a trip, but didn't still decide which route to follow and thus she asked you to help her determine summary price of passing **each of the roads** in XXX. Formally, for every pair of cities *a* and *b* (*a* < *b*), such that there is a road between *a* and *b* you are to find sum of products *ca*·*cb*. Will you help her?

**Input**

The first line of the input contains two integers *n* and *k* (3 ≤ *n* ≤ 100 000, 1 ≤ *k* ≤ *n*) — the number of cities in XXX and the number of capital cities among them.

The second line of the input contains *n* integers *c*1, *c*2, ..., *cn* (1 ≤ *ci* ≤ 10 000) — beauty values of the cities.

The third line of the input contains *k* distinct integers *id*1, *id*2, ..., *idk* (1 ≤ *idi* ≤ *n*) — indices of capital cities. Indices are given in ascending order.

**Output**

Print the only integer — summary price of passing each of the roads in XXX.

**Examples**

**input**

4 1  
2 3 1 2  
3

**output**

17

**input**

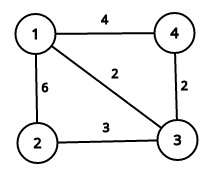
5 2  
3 5 2 2 4  
1 4

**output**

71

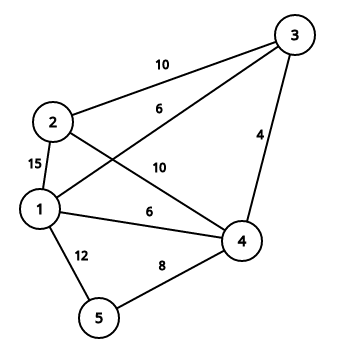
**Note**

This image describes first sample case:



It is easy to see that summary price is equal to 17.

This image describes second sample case:



It is easy to see that summary price is equal to 71.

A. Mishka and Game

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Mishka is a little polar bear. As known, little bears loves spending their free time playing dice for chocolates. Once in a wonderful sunny morning, walking around blocks of ice, Mishka met her friend Chris, and they started playing the game.

Rules of the game are very simple: at first number of rounds *n* is defined. In every round each of the players throws a cubical dice with distinct numbers from 1 to 6 written on its faces. Player, whose value after throwing the dice is greater, wins the round. In case if player dice values are equal, no one of them is a winner.

In average, player, who won most of the rounds, is the winner of the game. In case if two players won the same number of rounds, the result of the game is draw.

Mishka is still very little and can't count wins and losses, so she asked you to watch their game and determine its result. Please help her!

**Input**

The first line of the input contains single integer *n* *n* (1 ≤ *n* ≤ 100) — the number of game rounds.

The next *n* lines contains rounds description. *i*-th of them contains pair of integers *mi* and *ci* (1 ≤ *mi*,  *ci* ≤ 6) — values on dice upper face after Mishka's and Chris' throws in *i*-th round respectively.

**Output**

If Mishka is the winner of the game, print "Mishka" (without quotes) in the only line.

If Chris is the winner of the game, print "Chris" (without quotes) in the only line.

If the result of the game is draw, print "Friendship is magic!^^" (without quotes) in the only line.

**Examples**

**input**

3  
3 5  
2 1  
4 2

**output**

Mishka

**input**

2  
6 1  
1 6

**output**

Friendship is magic!^^

**input**

3  
1 5  
3 3  
2 2

**output**

Chris

**Note**

In the first sample case Mishka loses the first round, but wins second and third rounds and thus she is the winner of the game.

In the second sample case Mishka wins the first round, Chris wins the second round, and the game ends with draw with score 1:1.

In the third sample case Chris wins the first round, but there is no winner of the next two rounds. The winner of the game is Chris.

F. T-Shirts

time limit per test

4 seconds

memory limit per test

1024 megabytes

input

standard input

output

standard output

The big consignment of t-shirts goes on sale in the shop before the beginning of the spring. In all *n* types of t-shirts go on sale. The t-shirt of the *i*-th type has two integer parameters — *ci* and *qi*, where *ci* — is the price of the *i*-th type t-shirt, *qi* — is the quality of the *i*-th type t-shirt. It should be assumed that the unlimited number of t-shirts of each type goes on sale in the shop, but in general the quality is not concerned with the price.

As predicted, *k* customers will come to the shop within the next month, the *j*-th customer will get ready to spend up to *bj* on buying t-shirts.

All customers have the same strategy. First of all, the customer wants to buy the maximum possible number of the highest quality t-shirts, then to buy the maximum possible number of the highest quality t-shirts from residuary t-shirts and so on. At the same time among several same quality t-shirts the customer will buy one that is cheaper. The customers don't like the same t-shirts, so each customer will not buy more than one t-shirt of one type.

Determine the number of t-shirts which each customer will buy, if they use the described strategy. All customers act independently from each other, and the purchase of one does not affect the purchase of another.

**Input**

The first line contains the positive integer *n* (1 ≤ *n* ≤ 2·105) — the number of t-shirt types.

Each of the following *n* lines contains two integers *ci* and *qi* (1 ≤ *ci*, *qi* ≤ 109) — the price and the quality of the *i*-th type t-shirt.

The next line contains the positive integer *k* (1 ≤ *k* ≤ 2·105) — the number of the customers.

The next line contains *k* positive integers *b*1, *b*2, ..., *bk* (1 ≤ *bj* ≤ 109), where the *j*-th number is equal to the sum, which the *j*-th customer gets ready to spend on t-shirts.

**Output**

The first line of the input data should contain the sequence of *k* integers, where the *i*-th number should be equal to the number of t-shirts, which the *i*-th customer will buy.

**Examples**

**input**

3  
7 5  
3 5  
4 3  
2  
13 14

**output**

2 3

**input**

2  
100 500  
50 499  
4  
50 200 150 100

**output**

1 2 2 1

**Note**

In the first example the first customer will buy the t-shirt of the second type, then the t-shirt of the first type. He will spend 10 and will not be able to buy the t-shirt of the third type because it costs 4, and the customer will owe only 3. The second customer will buy all three t-shirts (at first, the t-shirt of the second type, then the t-shirt of the first type, and then the t-shirt of the third type). He will spend all money on it.

E. Analysis of Pathes in Functional Graph

time limit per test

2 seconds

memory limit per test

512 megabytes

input

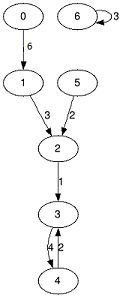
standard input

output

standard output

You are given a *functional graph*. It is a directed graph, in which from each vertex goes exactly one arc. The vertices are numerated from 0 to *n* - 1.

Graph is given as the array *f*0, *f*1, ..., *fn*- 1, where *fi* — the number of vertex to which goes the only arc from the vertex *i*. Besides you are given array with weights of the arcs *w*0, *w*1, ..., *wn*- 1, where *wi* — the arc weight from *i* to *fi*.

The graph from the first sample test.

Also you are given the integer *k* (the length of the path) and you need to find for each vertex two numbers *si* and *mi*, where:

* *si* — the sum of the weights of all arcs of the path with length equals to *k* which starts from the vertex *i*;
* *mi* — the minimal weight from all arcs on the path with length *k* which starts from the vertex *i*.

The length of the path is the number of arcs on this path.

**Input**

The first line contains two integers *n*, *k* (1 ≤ *n* ≤ 105, 1 ≤ *k* ≤ 1010). The second line contains the sequence *f*0, *f*1, ..., *fn*- 1 (0 ≤ *fi* < *n*) and the third — the sequence *w*0, *w*1, ..., *wn*- 1 (0 ≤ *wi* ≤ 108).

**Output**

Print *n* lines, the pair of integers *si*, *mi* in each line.

**Examples**

**input**

7 3  
1 2 3 4 3 2 6  
6 3 1 4 2 2 3

**output**

10 1  
8 1  
7 1  
10 2  
8 2  
7 1  
9 3

**input**

4 4  
0 1 2 3  
0 1 2 3

**output**

0 0  
4 1  
8 2  
12 3

**input**

5 3  
1 2 3 4 0  
4 1 2 14 3

**output**

7 1  
17 1  
19 2  
21 3  
8 1

D. Road to Post Office

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

Vasiliy has a car and he wants to get from home to the post office. The distance which he needs to pass equals to *d* kilometers.

Vasiliy's car is not new — it breaks after driven every *k* kilometers and Vasiliy needs *t* seconds to repair it. After repairing his car Vasiliy can drive again (but after *k* kilometers it will break again, and so on). In the beginning of the trip the car is just from repair station.

To drive one kilometer on car Vasiliy spends *a* seconds, to walk one kilometer on foot he needs *b* seconds (*a* < *b*).

Your task is to find minimal time after which Vasiliy will be able to reach the post office. Consider that in every moment of time Vasiliy can left his car and start to go on foot.

**Input**

The first line contains 5 positive integers *d*, *k*, *a*, *b*, *t* (1 ≤ *d* ≤ 1012; 1 ≤ *k*, *a*, *b*, *t* ≤ 106; *a* < *b*), where:

* *d* — the distance from home to the post office;
* *k* — the distance, which car is able to drive before breaking;
* *a* — the time, which Vasiliy spends to drive 1 kilometer on his car;
* *b* — the time, which Vasiliy spends to walk 1 kilometer on foot;
* *t* — the time, which Vasiliy spends to repair his car.

**Output**

Print the minimal time after which Vasiliy will be able to reach the post office.

**Examples**

**input**

5 2 1 4 10

**output**

14

**input**

5 2 1 4 5

**output**

13

**Note**

In the first example Vasiliy needs to drive the first 2 kilometers on the car (in 2 seconds) and then to walk on foot 3 kilometers (in 12 seconds). So the answer equals to 14 seconds.

In the second example Vasiliy needs to drive the first 2 kilometers on the car (in 2 seconds), then repair his car (in 5 seconds) and drive 2 kilometers more on the car (in 2 seconds). After that he needs to walk on foot 1 kilometer (in 4 seconds). So the answer equals to 13 seconds.

C. Cellular Network

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given *n* points on the straight line — the positions (*x*-coordinates) of the cities and *m* points on the same line — the positions (*x*-coordinates) of the cellular towers. All towers work in the same way — they provide cellular network for all cities, which are located at the distance which is no more than *r* from this tower.

Your task is to find minimal *r* that each city has been provided by cellular network, i.e. for each city there is at least one cellular tower at the distance which is no more than *r*.

If *r* = 0 then a tower provides cellular network only for the point where it is located. One tower can provide cellular network for any number of cities, but all these cities must be at the distance which is no more than *r* from this tower.

**Input**

The first line contains two positive integers *n* and *m* (1 ≤ *n*, *m* ≤ 105) — the number of cities and the number of cellular towers.

The second line contains a sequence of *n* integers *a*1, *a*2, ..., *an* ( - 109 ≤ *ai* ≤ 109) — the coordinates of cities. It is allowed that there are any number of cities in the same point. All coordinates *ai* are given in non-decreasing order.

The third line contains a sequence of *m* integers *b*1, *b*2, ..., *bm* ( - 109 ≤ *bj* ≤ 109) — the coordinates of cellular towers. It is allowed that there are any number of towers in the same point. All coordinates *bj* are given in non-decreasing order.

**Output**

Print minimal *r* so that each city will be covered by cellular network.

**Examples**

**input**

3 2  
-2 2 4  
-3 0

**output**

4

**input**

5 3  
1 5 10 14 17  
4 11 15

**output**

3

B. Powers of Two

time limit per test

3 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given *n* integers *a*1, *a*2, ..., *an*. Find the number of pairs of indexes *i*, *j* (*i* < *j*) that *ai* + *aj* is a power of 2 (i. e. some integer *x*exists so that *ai* + *aj* = 2*x*).

**Input**

The first line contains the single positive integer *n* (1 ≤ *n* ≤ 105) — the number of integers.

The second line contains *n* positive integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109).

**Output**

Print the number of pairs of indexes *i*, *j* (*i* < *j*) that *ai* + *aj* is a power of 2.

**Examples**

**input**

4  
7 3 2 1

**output**

2

**input**

3  
1 1 1

**output**

3

**Note**

In the first example the following pairs of indexes include in answer: (1, 4) and (2, 4).

In the second example all pairs of indexes (*i*, *j*) (where *i* < *j*) include in answer.

A. Maximum Increase

time limit per test

1 second

memory limit per test

256 megabytes

input

standard input

output

standard output

You are given array consisting of *n* integers. Your task is to find the maximum length of an increasing subarray of the given array.

A subarray is the sequence of consecutive elements of the array. Subarray is called increasing if each element of this subarray **strictly greater** than previous.

**Input**

The first line contains single positive integer *n* (1 ≤ *n* ≤ 105) — the number of integers.

The second line contains *n* positive integers *a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 109).

**Output**

Print the maximum length of an increasing subarray of the given array.

**Examples**

**input**

5  
1 7 2 11 15

**output**

3

**input**

6  
100 100 100 100 100 100

**output**

1

**input**

3  
1 2 3

**output**

3

C. They Are Everywhere

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

Sergei B., the young coach of Pokemons, has found the big house which consists of *n* flats ordered in a row from left to right. It is possible to enter each flat from the street. It is possible to go out from each flat. Also, each flat is connected with the flat to the left and the flat to the right. Flat number 1 is only connected with the flat number 2 and the flat number *n* is only connected with the flat number *n* - 1.

There is exactly one Pokemon of some type in each of these flats. Sergei B. asked residents of the house to let him enter their flats in order to catch Pokemons. After consulting the residents of the house decided to let Sergei B. enter one flat from the street, visit several flats and then go out from some flat. But they won't let him visit the same flat more than once.

Sergei B. was very pleased, and now he wants to visit as few flats as possible in order to collect Pokemons of all types that appear in this house. Your task is to help him and determine this minimum number of flats he has to visit.

**Input**

The first line contains the integer *n* (1 ≤ *n* ≤ 100 000) — the number of flats in the house.

The second line contains the row *s* with the length *n*, it consists of uppercase and lowercase letters of English alphabet, the *i*-th letter equals the type of Pokemon, which is in the flat number *i*.

**Output**

Print the minimum number of flats which Sergei B. should visit in order to catch Pokemons of all types which there are in the house.

**Examples**

**input**

3  
AaA

**output**

2

**input**

7  
bcAAcbc

**output**

3

**input**

6  
aaBCCe

**output**

5

**Note**

In the first test Sergei B. can begin, for example, from the flat number 1 and end in the flat number 2.

In the second test Sergei B. can begin, for example, from the flat number 4 and end in the flat number 6.

In the third test Sergei B. must begin from the flat number 2 and end in the flat number 6.