Sponsored by Telegram

VK Cup 2015 - Round 1

Statement is not available.

C. The Art of Dealing with ATM

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

ATMs of a well-known bank of a small country are arranged so that they can not give any amount of money requested by the user. Due to the limited size of the bill dispenser (the device that is directly giving money from an ATM) and some peculiarities of the ATM structure, you can get at most k bills from it, and the bills may be of at most two distinct denominations.

For example, if a country uses bills with denominations $10, 50, 100, 500, 1000$ and 5000 burles, then at $k = 20$ such ATM can give sums $100\,000$ burles and $96\,000$ burles, but it cannot give sums $99\,000$ and $101\,000$ burles.

Let's suppose that the country uses bills of n distinct denominations, and the ATM that you are using has an unlimited number of bills of each type. You know that during the day you will need to withdraw a certain amount of cash q times. You know that when the ATM has multiple ways to give money, it chooses the one which requires the minimum number of bills, or displays an error message if it cannot be done. Determine the result of each of the q of requests for cash withdrawal.

Input

The first line contains two integers n, k ($1 \le n \le 5000$, $1 \le k \le 20$).

The next line contains n space-separated integers a_i ($1 \le a_i \le 10^7$) — the denominations of the bills that are used in the country. Numbers a_i follow in the strictly increasing order.

The next line contains integer q ($1 \le q \le 20$) — the number of requests for cash withdrawal that you will make.

The next q lines contain numbers x_i ($1 \le x_i \le 2 \cdot 10^8$) — the sums of money in burles that you are going to withdraw from the ATM.

Output

For each request for cash withdrawal print on a single line the minimum number of bills it can be done, or print -1, if it is impossible to get the corresponding sum.

Sample test(s)

input

```
6 20
10 50 100 500 1000 5000
8
4200
100000
95000
96000
99000
10100
2015
9950
```

output

```
6
20
19
20
-1
3
-1
-1
```

input

```
5 2
1 2 3 5 8
8
1
3
5
7
9
11
13
15
```

output

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

D. Social Network

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarpus got an internship in one well-known social network. His test task is to count the number of unique users who have visited a social network during the day. Polycarpus was provided with information on all user requests for this time period. For each query, we know its time... and nothing else, because Polycarpus has already accidentally removed the user IDs corresponding to the requests from the database. Thus, it is now impossible to determine whether any two requests are made by the same person or by different people.

But wait, something is still known, because that day a record was achieved — M simultaneous users online! In addition, Polycarpus believes that if a user made a request at second s, then he was online for T seconds after that, that is, at seconds s, $s+1$, $s+2$, ..., $s+T-1$. So, the user's time online can be calculated as the union of time intervals of the form $[s, s+T-1]$ over all times s of requests from him.

Guided by these thoughts, Polycarpus wants to assign a user ID to each request so that:

- the number of different users online did not exceed M at any moment,
- at some second the number of distinct users online reached value M,
- the total number of users (the number of distinct identifiers) was as much as possible.

Help Polycarpus cope with the test.

Input

The first line contains three integers n, M and T ($1 \le n, M \le 20\,000$, $1 \le T \le 86400$) — the number of queries, the record number of online users and the time when the user was online after a query was sent. Next n lines contain the times of the queries in the format "hh:mm:ss", where hh are hours, mm are minutes, ss are seconds. The times of the queries follow in the non-decreasing order, some of them can coincide. It is guaranteed that all the times and even all the segments of type $[s, s+T-1]$ are within one 24-hour range (from 00:00:00 to 23:59:59).

Output

In the first line print number R — the largest possible number of distinct users. The following n lines should contain the user IDs for requests in the same order in which the requests are given in the input. User IDs must be integers from 1 to R. The requests of the same user must correspond to the same identifiers, the requests of distinct users must correspond to distinct identifiers. If there are multiple solutions, print any of them. If there is no solution, print "No solution" (without the quotes).

Sample test(s)

input

```
4 2 10
17:05:53
17:05:58
17:06:01
22:39:47
```

output

```
3
1
2
2
3
```

input

```
1 2 86400
00:00:00
```

output

```
No solution
```

Note

Consider the first sample. The user who sent the first request was online from 17:05:53 to 17:06:02, the user who sent the second request was online from 17:05:58 to 17:06:07, the user who sent the third request, was online from 17:06:01 to 17:06:10. Thus, these IDs cannot belong to three distinct users, because in that case all these users would be online, for example, at 17:06:01. That is impossible, because $M = 2$. That means that some two of these queries belonged to the same user. One of the correct variants is given in the answer to the sample. For it user 1 was online from 17:05:53 to 17:06:02, user 2 — from 17:05:58 to 17:06:10 (he sent the second and third queries), user 3 — from 22:39:47 to 22:39:56.

In the second sample there is only one query. So, only one user visited the network within the 24-hour period and there couldn't be two users online on the network simultaneously. (The time the user spent online is the union of time intervals for requests, so users who didn't send requests could not be online in the network.)

E. Rooks and Rectangles

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarpus has a chessboard of size $n \times m$, where k rooks are placed. Polycarpus hasn't yet invented the rules of the game he will play. However, he has already allocated q rectangular areas of special strategic importance on the board, they must be protected well. According to Polycarpus, a rectangular area of the board is well protected if all its vacant squares can be beaten by the rooks that stand on this area. The rooks on the rest of the board do not affect the area's defense. The position of the rooks is fixed and cannot be changed. We remind you that the the rook beats the squares located on the same vertical or horizontal line with it, if there are no other pieces between the square and the rook. Help Polycarpus determine whether all strategically important areas are protected.

Input

The first line contains four integers n, m, k and q $(1 \le n, m \le 100\,000,\ 1 \le k, q \le 200\,000)$ — the sizes of the board, the number of rooks and the number of strategically important sites. We will consider that the cells of the board are numbered by integers from 1 to n horizontally and from 1 to m vertically. Next k lines contain pairs of integers "x y", describing the positions of the rooks $(1 \le x \le n,\ 1 \le y \le m)$. It is guaranteed that all the rooks are in distinct squares. Next q lines describe the strategically important areas as groups of four integers "x_1 y_1 x_2 y_2" $(1 \le x_1 \le x_2 \le n$, $1 \le y_1 \le y_2 \le m)$. The corresponding rectangle area consists of cells (x, y), for which $x_1 \le x \le x_2$, $y_1 \le y \le y_2$. Strategically important areas can intersect of coincide.

Output

Print q lines. For each strategically important site print "YES" if it is well defended and "NO" otherwise.

Sample test(s)

input

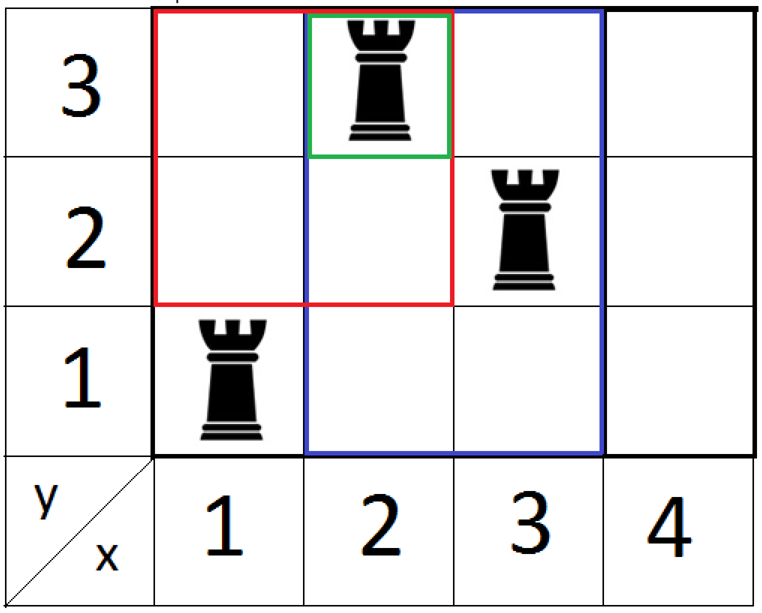
```
4 3 3 3
1 1
3 2
2 3
2 3 2 3
2 1 3 3
1 2 2 3
```

output

```
YES
YES
NO
```

Note

Picture to the sample:



For the last area the answer is "NO", because cell $(1, 2)$ cannot be hit by a rook.

F. And Yet Another Bracket Sequence

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarpus has a finite sequence of opening and closing brackets. In order not to fall asleep in a lecture, Polycarpus is having fun with his sequence. He is able to perform two operations:

- adding any bracket in any position (in the beginning, the end, or between any two existing brackets);
- cyclic shift — moving the last bracket from the end of the sequence to the beginning.

Polycarpus can apply any number of operations to his sequence and adding a cyclic shift in any order. As a result, he wants to get the correct bracket sequence of the minimum possible length. If there are several such sequences, Polycarpus is interested in the lexicographically smallest one. Help him find such a sequence.

A*correct bracket sequence* is a sequence of opening and closing brackets, from which you can get a correct arithmetic expression by adding characters "1" and "+" . Each opening bracket must correspond to a closed one. For example, the sequences "(())()", "()", "(()(()))" are correct and ")(", "(()" and "(()))(" are not.

The sequence a_1 a_2... a_n is lexicographically smaller than sequence b_1 b_2... b_n, if there is such number i from 1 to n, that$a_k = b_k$ for $1 \le k < i$ and $a_i < b_i$. Consider that "(" $<$ ")".

Input

The first line contains Polycarpus's sequence consisting of characters "(" and ")". The length of a line is from 1 to $1\,000\,000$.

Output

Print a correct bracket sequence of the minimum length that Polycarpus can obtain by his operations. If there are multiple such sequences, print the lexicographically minimum one.

Sample test(s)

input
```
()(())
```
output
```
(())()
```

input
```
()(
```
output
```
(())
```

Note

The sequence in the first example is already correct, but to get the lexicographically minimum answer, you need to perform four cyclic shift operations. In the second example you need to add a closing parenthesis between the second and third brackets and make a cyclic shift. You can first make the shift, and then add the bracket at the end.