CODEFORCES β
Sponsored by Telegram

Codeforces Round #436 (Div. 2)

A. Fair Game

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Petya and Vasya decided to play a game. They have n cards (n is an even number). A single integer is written on each card.

Before the game Petya will choose an integer and after that Vasya will choose another integer (**different** from the number that Petya chose). During the game each player takes all the cards with number he chose. For example, if Petya chose number 5 before the game he will take all cards on which 5 is written and if Vasya chose number 10 before the game he will take all cards on which 10 is written.

The game is considered fair if Petya and Vasya can take all n cards, and the number of cards each player gets is the same.

Determine whether Petya and Vasya can choose integer numbers before the game so that the game is fair.

Input

The first line contains a single integer n $(2 \le n \le 100)$ — number of cards. It is guaranteed that n is an even number.

The following n lines contain a sequence of integers $a_1, a_2, ..., a_n$ (one integer per line, $1 \le a_i \le 100)$ — numbers written on the n cards.

Output

If it is impossible for Petya and Vasya to choose numbers in such a way that the game will be fair, print "NO" (without quotes) in the first line. In this case you should not print anything more.

In the other case print "YES" (without quotes) in the first line. In the second line print two distinct integers — number that Petya should choose and the number that Vasya should choose to make the game fair. If there are several solutions, print any of them.

Examples

input

```
4
11
27
27
11
```

output

```
YES
11 27
```

input

```
2
6
6
```

output

```
NO
```

input

```
6
10
20
30
20
10
20
```

output

```
NO
```

input

```
6
1
1
2
2
3
3
```

output

```
NO
```

Note

In the first example the game will be fair if, for example, Petya chooses number 11, and Vasya chooses number 27. Then the will take all cards — Petya will take cards 1 and 4, and Vasya will take cards 2 and 3. Thus, each of them will take exactly two cards.

In the second example fair game is impossible because the numbers written on the cards are equal, but the numbers that Petya and Vasya should choose should be distinct.

In the third example it is impossible to take all cards. Petya and Vasya can take at most five cards — for example, Petya can choose number 10 and Vasya can choose number 20. But for the game to be fair it is necessary to take 6 cards.

B. Polycarp and Letters

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

Polycarp loves lowercase letters and dislikes uppercase ones. Once he got a string s consisting only of lowercase and uppercase Latin letters.

Let A be a set of positions in the string. Let's call it *pretty* if following conditions are met:

- letters on positions from A in the string are all distinct and lowercase;
- there are no uppercase letters in the string which are situated between positions from A (i.e. there is no such j that $s[j]$ is an uppercase letter, and $a_1 < j < a_2$ for some a_1 and a_2 from A).

Write a program that will determine the maximum number of elements in a *pretty* set of positions.

Input

The first line contains a single integer n ($1 \le n \le 200$) — length of string s.

The second line contains a string s consisting of lowercase and uppercase Latin letters.

Output

Print maximum number of elements in *pretty* set of positions for string s.

Examples

input
```
11
aaaaBaabAbA
```
output
```
2
```

input
```
12
zACaAbbaazzC
```
output
```
3
```

input
```
3
ABC
```
output
```
0
```

Note

In the first example the desired positions might be 6 and 8 or 7 and 8. Positions 6 and 7 contain letters 'a', position 8 contains letter 'b'. The pair of positions 1 and 8 is not suitable because there is an uppercase letter 'B' between these position.

In the second example desired positions can be 7, 8 and 11. There are other ways to choose *pretty* set consisting of three elements.

In the third example the given string s does not contain any lowercase letters, so the answer is 0.

C. Bus

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

A bus moves along the coordinate line Ox from the point $x = 0$ to the point $x = a$. After starting from the point $x = 0$, it reaches the point $x = a$, immediately turns back and then moves to the point $x = 0$. After returning to the point $x = 0$ it immediately goes back to the point $x = a$ and so on. Thus, the bus moves from $x = 0$ to $x = a$ and back. Moving from the point $x = 0$ to $x = a$ or from the point $x = a$ to $x = 0$ is called a *bus journey*. In total, the bus must make k journeys.

The petrol tank of the bus can hold b liters of gasoline. To pass a single unit of distance the bus needs to spend exactly one liter of gasoline. The bus starts its first journey with a full petrol tank.

There is a gas station in point $x = f$. This point is between points $x = 0$ and $x = a$. There are no other gas stations on the bus route. While passing by a gas station in either direction the bus can stop and completely refuel its tank. Thus, after stopping to refuel the tank will contain b liters of gasoline.

What is the minimum number of times the bus needs to refuel at the point $x = f$ to make k journeys? The first journey starts in the point $x = 0$.

Input

The first line contains four integers a, b, f, k ($0 < f < a \le 10^6$, $1 \le b \le 10^9$, $1 \le k \le 10^4$) — the endpoint of the first bus journey, the capacity of the fuel tank of the bus, the point where the gas station is located, and the required number of journeys.

Output

Print the minimum number of times the bus needs to refuel to make k journeys. If it is impossible for the bus to make k journeys, print `-1`.

Examples

input
```
6 9 2 4
```
output
```
4
```

input
```
6 10 2 4
```
output
```
2
```

input
```
6 5 4 3
```
output
```
-1
```

Note

In the first example the bus needs to refuel during each journey.

In the second example the bus can pass 10 units of distance without refueling. So the bus makes the whole first journey, passes 4 units of the distance of the second journey and arrives at the point with the gas station. Then it can refuel its tank, finish the second journey and pass 2 units of distance from the third journey. In this case, it will again arrive at the point with the gas station. Further, he can refill the tank up to 10 liters to finish the third journey and ride all the way of the fourth journey. At the end of the journey the tank will be empty.

In the third example the bus can not make all 3 journeys because if it refuels during the second journey, the tanks will contain only 5 liters of gasoline, but the bus needs to pass 8 units of distance until next refueling.

D. Make a Permutation!

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

Ivan has an array consisting of n elements. Each of the elements is an integer from 1 to n.

Recently Ivan learned about permutations and their lexicographical order. Now he wants to change (replace) *minimum* number of elements in his array in such a way that his array becomes a *permutation* (i.e. each of the integers from 1 to n was encountered in his array exactly once). If there are multiple ways to do it he wants to find the *lexicographically minimal* permutation among them.

Thus minimizing the number of changes has the first priority, lexicographical minimizing has the second priority.

In order to determine which of the two permutations is lexicographically smaller, we compare their first elements. If they are equal — compare the second, and so on. If we have two permutations x and y, then x is lexicographically smaller if $x_i < y_i$, where i is the first index in which the permutations x and y differ.

Determine the array Ivan will obtain after performing all the changes.

Input

The first line contains an single integer n ($2 \le n \le 200\,000$) — the number of elements in Ivan's array.

The second line contains a sequence of integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le n$) — the description of Ivan's array.

Output

In the first line print q — the minimum number of elements that need to be changed in Ivan's array in order to make his array a permutation. In the second line, print the *lexicographically minimal* permutation which can be obtained from array with q changes.

Examples

input

```
4
3 2 2 3
```

output

```
2
1 2 4 3
```

input

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

output

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

input

```
10
6 8 4 6 7 1 6 3 4 5
```

output

```
3
2 8 4 6 7 1 9 3 10 5
```

Note

In the first example Ivan needs to replace number three in position 1 with number one, and number two in position 3 with number four. Then he will get a permutation `[1, 2, 4, 3]` with only two changed numbers — this permutation is lexicographically minimal among all suitable.

In the second example Ivan does not need to change anything because his array already is a permutation.

E. Fire

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

Polycarp is in really serious trouble — his house is on fire! It's time to save the most valuable items. Polycarp estimated that it would take t_i seconds to save i-th item. In addition, for each item, he estimated the value of d_i — the moment after which the item i will be completely burned and will no longer be valuable for him at all. In particular, if $t_i \geq d_i$, then i-th item cannot be saved.

Given the values p_i for each of the items, find a set of items that Polycarp can save such that the total value of this items is maximum possible. Polycarp saves the items one after another. For example, if he takes item a first, and then item b, then the item a will be saved in t_a seconds, and the item b — in $t_a + t_b$ seconds after fire started.

Input

The first line contains a single integer n $(1 \leq n \leq 100)$ — the number of items in Polycarp's house.

Each of the following n lines contains three integers t_i, d_i, p_i $(1 \leq t_i \leq 20,\ 1 \leq d_i \leq 2\,000,\ 1 \leq p_i \leq 20)$ — the time needed to save the item i, the time after which the item i will burn completely and the value of item i.

Output

In the first line print the maximum possible total value of the set of saved items. In the second line print one integer m — the number of items in the desired set. In the third line print m distinct integers — numbers of the saved items *in the order Polycarp saves them*. Items are 1-indexed in the same order in which they appear in the input. If there are several answers, print any of them.

Examples

input

```
3
3 7 4
2 6 5
3 7 6
```

output

```
11
2
2 3
```

input

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

output

```
1
1
1
```

Note

In the first example Polycarp will have time to save any two items, but in order to maximize the total value of the saved items, he must save the second and the third item. For example, he can firstly save the third item in 3 seconds, and then save the second item in another 2 seconds. Thus, the total value of the saved items will be $6 + 5 = 11$.

In the second example Polycarp can save only the first item, since even if he immediately starts saving the second item, he can save it in 3 seconds, but this item will already be completely burned by this time.

F. Cities Excursions

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

There are n cities in Berland. Some pairs of them are connected with m directed roads. One can use only these roads to move from one city to another. There are no roads that connect a city to itself. For each pair of cities (x, y) there is at most one road from x to y.

A path from city s to city t is a sequence of cities $p_1, p_2, \ldots, p_k$, where $p_1 = s$, $p_k = t$, and there is a road from city p_i to city p_{i+1} for each i from 1 to $k - 1$. The path can pass multiple times through each city except t. It can't pass through t more than once.

A path p from s to t is *ideal* if it is the lexicographically minimal such path. In other words, p is *ideal* path from s to t if for any other path q from s to t $p_i < q_i$, where i is the minimum integer such that $p_i \neq q_i$.

There is a tourist agency in the country that offers q unusual excursions: the j-th excursion starts at city s_j and ends in city t_j.

For each pair s_j, t_j help the agency to study the ideal path from s_j to t_j. Note that it is possible that there is no ideal path from s_j to t_j. This is possible due to two reasons:

- there is no path from s_j to t_j;
- there are paths from s_j to t_j, but for every such path p there is another path q from s_j to t_j, such that $p_i > q_i$, where i is the minimum integer for which $p_i \neq q_i$.

The agency would like to know for the ideal path from s_j to t_j the k_j-th city in that path (on the way from s_j to t_j).

For each triple s_j, t_j, k_j $(1 \le j \le q)$ find if there is an ideal path from s_j to t_j and print the k_j-th city in that path, if there is any.

Input

The first line contains three integers n, m and q $(2 \le n \le 3000, 0 \le m \le 3000,\ 1 \le q \le 4 \cdot 10^5)$ — the number of cities, the number of roads and the number of excursions.

Each of the next m lines contains two integers x_i and y_i $(1 \le x_i, y_i \le n,\ x_i \neq y_i)$, denoting that the i-th road goes from city x_i to city y_i. All roads are one-directional. There can't be more than one road in each direction between two cities.

Each of the next q lines contains three integers s_j, t_j and k_j $(1 \le s_j, t_j \le n,\ s_j \neq t_j,\ 1 \le k_j \le 3000)$.

Output

In the j-th line print the city that is the k_j-th in the ideal path from s_j to t_j. If there is no ideal path from s_j to t_j, or the integer k_j is greater than the length of this path, print the string '`-1`' (without quotes) in the j-th line.

Example

input

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

output

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