Codeforces Round #360 (Div. 1)

A. NP-Hard Problem

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

Recently, Pari and Arya did some research about NP-Hard problems and they found the *minimum vertex cover* problem very interesting.

Suppose the graph G is given. Subset A of its vertices is called a *vertex cover* of this graph, if for each edge uv there is at least one endpoint of it in this set, i.e. or (or both).

Pari and Arya have won a great undirected graph as an award in a team contest. Now they have to split it in two parts, but both of them want their parts of the graph to be a vertex cover.

They have agreed to give you their graph and you need to find two **disjoint** subsets of its vertices A and B, such that both A and B are vertex cover or claim it's impossible. Each vertex should be given to no more than one of the friends (or you can even keep it for yourself).

Input

The first line of the input contains two integers n and m ($2 \le n \le 100\,000$, $1 \le m \le 100\,000$) — the number of vertices and the number of edges in the prize graph, respectively.

Each of the next m lines contains a pair of integers u_i and v_i ($1 \le u_i, v_i \le n$), denoting an undirected edge between u_i and v_i. It's guaranteed the graph won't contain any self-loops or multiple edges.

Output

If it's impossible to split the graph between Pari and Arya as they expect, print "-1" (without quotes).

If there are two disjoint sets of vertices, such that both sets are vertex cover, print their descriptions. Each description must contain two lines. The first line contains a single integer k denoting the number of vertices in that vertex cover, and the second line contains k integers — the indices of vertices. Note that because of $m \ge 1$, vertex cover cannot be empty.

Examples

input
4 2 1 2 2 3
output
1 2 2 1 3

input
3 3 1 2 2 3 1 3
output
-1

Note

In the first sample, you can give the vertex number 2 to Arya and vertices numbered 1 and 3 to Pari and keep vertex number 4 for yourself (or give it someone, if you wish).

In the second sample, there is no way to satisfy both Pari and Arya.

B. Remainders Game

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

Today Pari and Arya are playing a game called Remainders.

Pari chooses two positive integer x and k, and tells Arya k but not x. Arya have to find the value . There are n ancient numbers $c_1, c_2, ..., c_n$ and Pari has to tell Arya if Arya wants. Given k and the ancient values, tell us if Arya has a winning strategy independent of value of x or not. Formally, is it true that Arya can understand the value for any positive integer x?

Note, that means the remainder of x after dividing it by y.

Input

The first line of the input contains two integers n and k ($1 \le n,\ k \le 1\,000\,000$) — the number of ancient integers and value k that is chosen by Pari.

The second line contains n integers $c_1, c_2, ..., c_n$ ($1 \le c_i \le 1\,000\,000$).

Output

Print "Yes" (without quotes) if Arya has a winning strategy independent of value of x, or "No" (without quotes) otherwise.

Examples

input
4 5 2 3 5 12
output
Yes

input
2 7 2 3
output
No

Note

In the first sample, Arya can understand because 5 is one of the ancient numbers.

In the second sample, Arya can't be sure what is. For example 1 and 7 have the same remainders after dividing by 2 and 3, but they differ in remainders after dividing by 7.

C. The Values You Can Make

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

Pari wants to buy an expensive chocolate from Arya. She has n coins, the value of the i-th coin is c_i. The price of the chocolate is k, so Pari will take a subset of her coins with sum equal to k and give it to Arya.

Looking at her coins, a question came to her mind: after giving the coins to Arya, what values does Arya can make with them? She is jealous and she doesn't want Arya to make a lot of values. So she wants to know all the values x, such that Arya will be able to make x using some subset of coins with the sum k.

Formally, Pari wants to know the values x such that there exists a subset of coins with the sum k such that some subset of this subset has the sum x, i.e. there is exists some way to pay for the chocolate, such that Arya will be able to make the sum x using these coins.

Input

The first line contains two integers n and k $(1 \le n, k \le 500)$ — the number of coins and the price of the chocolate, respectively.

Next line will contain n integers $c_1, c_2, ..., c_n$ $(1 \le c_i \le 500)$ — the values of Pari's coins.

It's guaranteed that one can make value k using these coins.

Output

First line of the output must contain a single integer q— the number of suitable values x. Then print q integers in ascending order — the values that Arya can make for some subset of coins of Pari that pays for the chocolate.

Examples

input
6 18 5 6 1 10 12 2
output
16 0 1 2 3 5 6 7 8 10 11 12 13 15 16 17 18

input
3 50 25 25 50
output
3 0 25 50

D. Dividing Kingdom II

time limit per test: 6 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Long time ago, there was a great kingdom and it was being ruled by The Great Arya and Pari The Great. These two had some problems about the numbers they like, so they decided to divide the great kingdom between themselves.

The great kingdom consisted of n cities numbered from 1 to n and m bidirectional roads between these cities, numbered from 1 to m. The i-th road had length equal to w_i. The Great Arya and Pari The Great were discussing about destructing some prefix (all road with numbers less than some x) and suffix (all roads with numbers greater than some x) of the roads so there will remain only the roads with numbers $l, l+1, ..., r$ - 1 and r.

After that they will divide the great kingdom into two pieces (with each city belonging to exactly one piece) such that the *hardness* of the division is **minimized**. The hardness of a division is the **maximum length** of a road such that its both endpoints are in the same piece of the kingdom. In case there is no such road, the hardness of the division is considered to be equal to -1.

Historians found the map of the great kingdom, and they have q guesses about the l and r chosen by those great rulers. Given these data, for each guess l_i and r_i print the minimum possible hardness of the division of the kingdom.

Input

The first line of the input contains three integers n, m and q ($1 \le n, q \le 1000,$) — the number of cities and roads in the great kingdom, and the number of guesses, respectively.

The i-th line of the following m lines contains three integers u_i, v_i and w_i ($1 \le u_i, v_i \le n, 0 \le w_i \le 10^9$), denoting the road number i connects cities u_i and v_i and its length is equal w_i. It's guaranteed that no road connects the city to itself and no pair of cities is connected by more than one road.

Each of the next q lines contains a pair of integers l_i and r_i ($1 \le l_i \le r_i \le m$) — a guess from the historians about the remaining roads in the kingdom.

Output

For each guess print the minimum possible hardness of the division in described scenario.

Example

input
5 6 5 5 4 86 5 1 0 1 3 38 2 1 33 2 4 28 2 3 40 3 5 2 6 1 3 2 3 1 6

output
-1 33 -1 -1 33

E. TOF

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

Today Pari gave Arya a cool graph problem. Arya wrote a non-optimal solution for it, because he believes in his ability to optimize non-optimal solutions. In addition to being non-optimal, his code was buggy and he tried a lot to optimize it, so the code also became dirty! He keeps getting Time Limit Exceeds and he is disappointed. Suddenly a bright idea came to his mind!

Here is how his dirty code looks like:

```
dfs(v)
{
     set count[v] = count[v] + 1
     if(count[v] < 1000)
     {
          foreach u in neighbors[v]
          {
               if(visited[u] is equal to false)
               {
                    dfs(u)
               }
               break
          }
     }
     set visited[v] = true
}

main()
{
     input the digraph()
     TOF()
     foreach 1<=i<=n
     {
          set count[i] = 0 , visited[i] = false
     }
     foreach 1 <= v <= n
     {
          if(visited[v] is equal to false)
          {
               dfs(v)
          }
     }
     ... // And do something cool and magical but we can't tell you what!
}
```

He asks you to write the *TOF* function in order to optimize the running time of the code with minimizing the number of calls of the *dfs* function. The input is a directed graph and in the *TOF* function you have to rearrange the edges of the graph in the list *neighbors* for each vertex. The number of calls of *dfs* function depends on the arrangement of *neighbors* of each vertex.

Input

The first line of the input contains two integers n and m $(1 \leq n, m \leq 5000)$ — the number of vertices and then number of directed edges in the input graph.

Each of the next m lines contains a pair of integers u_i and v_i $(1 \leq u_i, v_i \leq n)$, meaning there is a directed edge in the input graph.

You may assume that the graph won't contain any self-loops and there is at most one edge between any unordered pair of vertices.

Output

Print a single integer — the minimum possible number of *dfs* calls that can be achieved with permuting the edges.

Examples

input
3 3 1 2 2 3 3 1
output

```
2998
```

input

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

output

```
3001
```