Problem B

We have a network of computers, some of which are clients and servers. When some of the clients want to accomplish a task, each of them has to pick a distinct server to communicate with. Furthermore, as the amount of data that needs to be transferred between the servers and clients is huge, each of them has to pick a path through the network along which it sends data, and the paths for different clients cannot pass through the same computer.

This is not always possible. Hence, each client will also tell us how much it is willing to pay to accomplish its task. We want to select a subset of the clients that can work on their tasks simultaneously so that we earn as much as possible.

More precisely, let G be a graph and let C and S be disjoint sets of vertices of G. A set $C' \subseteq C$ is *viable* if G contains $|C'|$ pairwise vertex-disjoint paths starting in C' and ending in S (the paths can pass through other vertices of C). Given an assignment of weights to vertices of C, find a viable subset of C whose total weight is maximum.

Input and output

The first line contains integers $n, m \leq 100\,000$, the number of vertices and edges of G, and integers $1 \leq c, s \leq 100$ such that $c + s \leq n$, the number of clients and servers. The vertices of G are numbered from 1 to n, the vertices $1, \ldots, c$ are clients, the vertices $n - s + 1, \ldots, s$ are servers. Each of the following m lines contains three integers u and v, where $1 \leq u < v \leq n$; this indicates G contains an edge between u and v. You can assume that G contains at most one edge between any two vertices.

Each of the following lines describes a scenario. It contains c pairwise different positive integers smaller than $10\,000$, indicating how much are the clients $1, \ldots, c$ willing to pay to accomplish a task. For each of them, output a viable subset of clients whose total weight is maximum; the numbers of selected clients should be listed in increasing order and separated by spaces.

Example

Input:

```
5 6 3 2
1 2
2 4
2 5
4 5
3 4
3 5
5 10 1
10 5 1
```

Output:

```
2 3
1 3
```