Programming task 1

Write a program which determines the order of execution of mutually dependent tasks on a processor with one execution unit.

Model

We assume working on a computer with one processor that contains one execution unit (so no parallelism of computation is possible).

We are given a list of **tasks**, numbered 1, …, *n*, which are supposed to be executed on such computer. We assume that executing a single task happens **contiguously** in time, not interrupted by work on any other tasks.

We are also given additional information about the tasks:

- We're given a description of cross-task dependencies.
  When task *j* **depends** on task *i*, this means that executing task *j* can be started only once task *i* has been completed. Every task can depend on an aribtrary number of other tasks (0, 1, or more).

- For each task *k*, we're given its **priority** p_k, which is an integer number. If the processor has at some moment a choice between several tasks, it will always pick the one with top priority, which is the one for which the priority number p_k has **lowest** value.

Your goal is to determine the **order** in which the processor will execute the specified tasks.

Input

The program should read the data from the standard input (also known as: the console, the terminal, the command line).

The input describes the tasks in the following format:

- The first line contains one positive integer *n* – the number of tasks;
- Each of the following *n* lines contain one positive integer; these number are pairwise distinct and specify the priorities of tasks 1, 2, …, *n* respectively;
- The following line contains one non-negative integer *d* – the number of known dependencies between the tasks;
- Each of the following *d* lines describe one cross-task dependency by two integers (in the range between 1 and *n*), separated by one whitespace. The notation "*i j*" means that task *j* depends on task *i*.

Output

The program should print to the output *n* lines, each containing one integer, which describe the order of executing the tasks by the processor. In the *k*-th line, it should print out the number of tasks which will be executed as the *k*-th.

Example

For the input:

```
6
7
15
20
10
22
3
5
3 1
2 5
2 4
4 5
3 6
```

we're given 6 tasks, with priorities 7, 15, 20, 10, 22, 3, respectively, and the following dependencies:

- task 1 depends on task 3,
- task 4 depends on task 2,
- task 5 depends on tasks 2 and 4,
- task 6 depends on task 3.

For this setup, the execution of tasks will proceed as follows:

- Initially, the processor can choose between tasks 2 and 3 (as all other tasks have initially some unsatisfied dependencies). Their priorities are, respectively, 15 and 20, which means that, according to priority, the processor will pick and execute task **2**.
- Once task 2 is completed, task 4 gets unblocked. (On the other hand, task 5 remains blocked, as it also depends on task 4 which has not been done yet). So, now, the processor can choose between tasks 3 and 4, with priorities respectively 20 and 10. Given this, we'll now execute task **4**.
- Completing task 4 unblocks task 5. We now have a choice between tasks 3 and 5; according to priority, we pick task **3.**
- Completing task 3 unblocks tasks 1 and 6. We now have a choice between tasks 1, 5 and 6; guided by their priorities, we will execute them in the following order: **6**, **1**, **5**.

Altogether, the processor will execute the tasks in the order **2**, **4**, **3**, **6**, **1**, **5**. This means that the program should print out the following output:

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

Assumptions

For your technical convenience, assume that:

- The input has the correct format (as specified in the "Input" section above).

- The number n does not exceed 10 000, and d does not exceed n^2.
- The priority values do not exceed 1 000 000.
- The priority values for different tasks are pairwise distinct.
- There are no cyclic dependencies between tasks (like: 1 depending on 2, 2 depending on 3, 3 depending back on 1). Such dependencies would make it impossible to execute all tasks. (On the other hand, it can be proved that if there are no cyclic dependencies, then a set of tasks can be executed within the model described here).

In this task, I **do not restrict** your choice of programming language – you can choose Python (which I recommend!), Java, C++, or any other language.

Bonus (+20% of points)

You can score extra points for basing your code on Kahn's algorithm (I recommend reading its description here) – so that its computational efficiency is ensured.

Submitting solutions

Solutions should be submitted by **May 9** in Edux.