CS 330 – Topological Ordering

deadline: Sep. 27, 2019, 11:59 pm

In this exercise you will create an implementation (in Python 3 or Java) of the topological ordering algorithm given in Section 3.6 of *Algorithm Design.* You must implement an efficient version of the high level algorithm given in the textbooks. This means you will need to efficiently find nodes with no incoming edges. See the lecture slides for addtional details. Your code must run in $\Theta(n+m)$ time, where n is the number of vertices and m is the number of edges. We've set rough upper bounds on runtime through the autograder; if your code exceeds these, you will see the limit and your code's runtime.

Submission Format

On Gradescope submit a file (either `topological_ordering.py` or `topological_ordering.java`, names must match exactly) that reads a text file `input` and creates a text file `output` containing your solution. The autograder will put your code in the same directory as `input` and call it from the command line with either

```
> python3 topological_ordering.py
```

or, if you submit Java code,

```
> javac topological_ordering.java
> java topological_ordering
```

Your code must create and write `output` to the same directory. If you use an IDE we suggest you also test your code from the command line.

Input

The file `input` contains $n+1$ rows, specifying a directed acyclic graph in the adjacency list representation. You do not need to verify that the input graph is acyclic. The first line will be the integer n, the number of vertices. The vertices will be indexed $0, 1, \dots, n-1$. Every line after that will specify the destination of the edges coming from that vertex, as comma-separated indices. Vertices with no outgoing edges will have a '`-`' on their line.

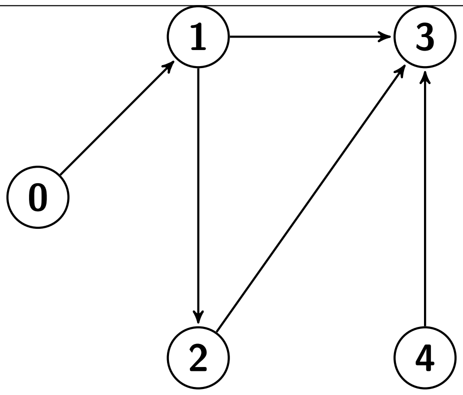
The lines of `input` will contain the following information:

1. n, the number of vertices.
2. Comma-separated list of neighbors: all v such that there is an edge $(0, v)$.
3. Comma-separated list of neighbors: all v such that there is an edge $(1, v)$.
4. ...
5. Comma-separated list of neighbors: all v such that there is an edge $(n-1, v)$.

We have uploaded files corresponding to the visible test cases on Gradescope. They are to help you test and debug locally. Note: passing these test cases does not guarantee your code is correct!

Output

The output must be a text file `output` giving a valid topological ordering. Starting with the first line, write on each line the index of a vertex. If there is an edge (u, v) in the graph, then u must appear on a higher line than v.



Example

Consider the the following DAG and its associated input file:

```
input:

5
1
2,3
3
-
3
```

One valid topological ordering is given by:

```
output:

0
1
2
4
3
```