Assignment overview:

I hope that you are all finding ways to deal with the diverse challenges we all face due to COVID-19. One way some people are coping with isolation is by doing projects around their homes. This lab may help with that in a small, geeky way.

Problem: Suppose you have a list of household to-do items, but for some of them starting one task requires finishing another task first. If you have many tasks with dependencies on others, it may be difficult to determine a single prioritized order in which all the tasks can be completed.

You can solve this problem with Topological Sorting. Represent your to-do list as a graph where every task is a vertex and every dependency is a directed edge from one task to another. Then a Topological Sort Ordering of the vertices will give a priority order for the tasks.

Submission information

Due date: As shown on Learning Hub. Late assignments will not be graded.

What to submit: Just your java code

You may (and should!) discuss the lab and coding techniques with your classmates, but all of the code you submit to Learning Hub must be your own.

Summary of requirements:

This lab is worth 15 marks, broken down as follows:

1. Implement a class for directed graphs (9 marks total)
    a. Constructor with filename argument (String) (3 marks)
    b. Public method to display the graph (3 marks)
    c. (3 marks for any other implementation details)
2. Implement a topological sort algorithm (6 marks)

Detailed requirements about the graph class

((Req1)) Your class must be named “DirectedGraph”.

((Req2)) The DirectedGraph class must have a constructor that takes a single String argument – a file name. This constructor will read information about vertices and edges of a graph from the given file and initialize the DirectedGraph object to represent that graph. See below for a detailed description of the data file(s).

((Req3)) The class must have a public method called “PrintGraph” which prints the graph to the console output. The printed output must include (1) a list of the names of the vertices and (2) some indication of the edges in the graph. The latter could be a plain list of the edges themselves (pairs of vertices), or it could be the data items from an adjacency matrix or adjacency list.

For example, here are two different but both acceptable outputs of the same graph (filename “graph14.txt”):

```
Graph output possibility 1
Vertices: a b c d e
Edges: (a,b) (a,c) (b,c) (b,e) (c,e) (d,a) (d,b) (d,c) (d,e)

Graph output possibility 2
Vertex labels:
0:a 1:b 2:c 3:d 4:e
Adjacency matrix:
0 1 1 0 0
0 0 1 0 1
0 0 0 0 1
1 1 1 0 1
0 0 0 0 0
```

Your graph class may use any internal data representation and any private or public class members (properties and methods) that you wish. You may use any built-in Java data types (arrays, lists, sets, maps, …) that you find helpful.

Exception: I don't know if any Java libraries happen to include Directed Graphs or Topological Sorting, but if they do, you may not use those!

Details about the Topological Sort

((Req4)) Write a function called topoSort(G) which takes a graph as an argument and returns an array of Strings. The returned array will contain the labels of the vertices of graph G in a valid Topological Sort Order.

You may choose either of the two topological sort algorithms that we have learned – one based on DFS, and the other a decrease-by-1 and conquer algorithm that involves removing one vertex from the graph per iteration.

topoSort() is a standalone function. It is *not* a method of the graph class.

The topoSort() function must not perform any other input or output.

Your graph class may include any public or private members (properties and methods) that you wish to have in order to facilitate the working of this function. There are no points for programming style here, but please don't take this as an invitation to go too crazy. Better coding style == easier to develop, easier to debug, and (perhaps most importantly) easier to *grade*.

Input data file specifications

The first line of the data file contains an integer N, the number of vertices in the graph.

The next N lines each contain a string which is the label of one vertex. The labels may contain spaces. The first label is for vertex 0, the next for vertex 1, and so on through vertex N-1.

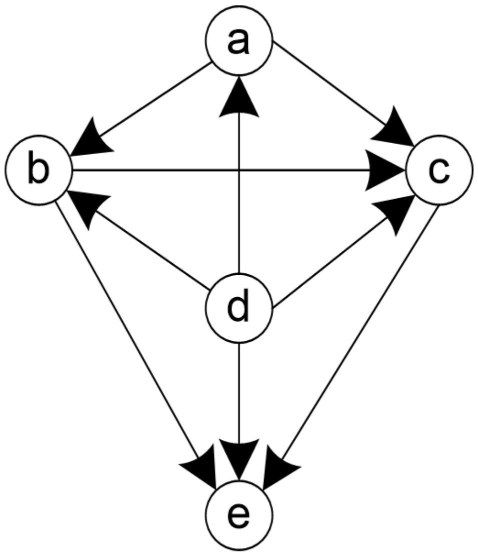
The N+2 line of the data file contains another integer K which is the number of edges in the graph.

The next (and last) K lines each contain two integers *i*,*j* separated by space. These values represent a directed edge from vertex *i* to vertex *j*.

Here is a sample data file ("graph14.txt"):

```
5
a
b
c
d
e
9
0 1
0 2
1 2
1 4
2 4
3 0
3 1
3 2
3 4
```

And here is the graph that is represented by this data file:



Output

A valid Topological Sort Order for the above graph, i.e. the list of strings that would be returned by the topoSort() function, would be:

d, a, b, c, e

PFG

One (1) PFG for the first person to tell me (on Slack) why I named this data file "graph14.txt". It is *not* because the number of vertices+edges 5+9=14, although that coincidence amuses me.