Programming Assignment #5: ConstrainedTopoSort

COP 3503, Spring 2017

Due: Sunday, March 26, *before* 11:59 PM

Abstract

In this assignment, you will determine whether an arbitrary directed graph has a valid topological sort in which some vertex, *x*, comes before some other vertex, *y*.

You will gain experience reading graphs from an input file, representing them computationally, and writing graph theory algorithms. You will also solidify your understanding of topological sorts, sharpen your problem solving skills, and get some practice at being clever, because your solution to this problem must be $O(n^2)$. In coming up with a solution, I recommend focusing first on developing a working algorithm, and *then* analyzing its runtime. (Focusing *too* much on the runtime restriction might cloud your thought as you cook up a solution to this problem.)

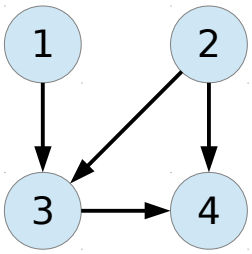
If you use any code that I have given you so far in class, you should probably include a comment to give me credit. The intellectually curious student will, of course, try to write the whole program from scratch.

Deliverables

ConstrainedTopoSort.java

1. Problem Statement

Given a directed graph, G, and two integers, *x* and *y*, determine whether G has a valid topological sort in which vertex *x* comes before vertex *y*. (Notice that the problem does **not** ask whether *x* comes *directly* before *y*.) For example:



G_1

In G_1, there is a valid topological sort in which vertex 2 comes before vertex 1 (**2, 1, 3, 4**). There is also a valid topological sort in which vertex 1 comes before vertex 2 (**1, 2, 3, 4**). Both of those are also valid topological sorts in which vertex 2 comes before vertex 4. (Notice that vertex 2 does not have to come *directly* before vertex 4.) However, there is no valid topological sort in which vertex 4 comes before vertex 1.

2. Input File Format

Each input file contains a single digraph. The first line contains a single integer, $n \geq 2$, indicating the number of vertices in the graph. (Vertices in these graphs are numbered 1 through *n*.) The following *n* lines are the adjacency lists for each successive vertex in the graph. Each adjacency list begins with a single non-negative integer, *k*, indicating the number of vertices that follow. The list of vertices that follows will contain *k* distinct integers (i.e., no repeats) on the range 1 through *n*. For example, the following text file corresponds to the graph G_1 that is pictured above:

g1.txt

```
4
1 3
2 3 4
1 4
0
```

3. Method and Class Requirements

Implement the following methods in a class named `ConstrainedTopoSort`.

`public ConstrainedTopoSort(String filename)`

This constructor opens the file named *filename* and reads the graph it contains into either an adjacency matrix or adjacency list. We will process multiple *xy* queries for this graph, but we only want to load it into memory once. This method should throw exceptions as necessary.

`public boolean hasConstrainedTopoSort(int x, int y)`

Given integers *x* and *y* such that $1 \leq x \leq n$ and $1 \leq y \leq n$, if this graph has a valid topological sort in which vertex *x* precedes vertex *y*, return *true*. Otherwise, return *false*. Do this in **O(n^2)** time.

`public static double difficultyRating()`

Return a double on the range 1.0 (ridiculously easy) to 5.0 (insanely difficult).

`public static double hoursSpent()`

Return an estimate (greater than zero) of the number of hours you spent on this assignment.

Runtime Requirement: Please note that you must implement a solution that is **O(n^2)**, where $n = |V|$. Recall from our formal definition of Big-Oh that a faster solution is still considered $O(n^2)$.

4. Grading Criteria and Miscellaneous Requirements

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

80%	program passes test cases
10%	`difficultyRating()` and `hoursSpent()` return doubles in the specified ranges
10%	adequate comments and whitespace

Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

Important! You might want to remove `main()` and then double check that your program compiles without it before submitting. Including a `main()` method can cause compilation issues if it includes references to home-brewed classes that you are not submitting with the assignment. Please remove.

Important! Your program should not print anything to the screen. Extraneous output is disruptive to the grading process and will result in severe point deductions. Please do not print to the screen.

Important! Please do not create a java `package`. Articulating a `package` in your source code could prevent it from compiling with our test cases, resulting in severe point deductions.

Important! Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Similarly, failing to write any one of the required methods, or failing to make them `public` and/or `static` (as appropriate) may cause test case failure. Please double check your work!

Input specifications are a contract. We promise that we will work within the confines of the problem statement when creating the test cases that we'll use for grading. Please reflect carefully on the kinds of edge cases that might cause unusual behaviors for any of the methods you're implementing, and be sure to write your own test cases to check them.

Start early! Work hard! Ask questions! Good luck!