Programming Assignment 4

CSCD320 Algorithms

Eastern Washington University, Spokane, Washington

Please follow these rules strictly:

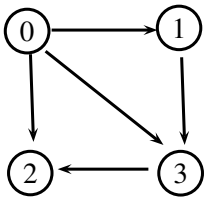
1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own work, without referring to anybody else's solution.
2. No one should give his/her code to anyone else.
3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
4. Every source code file must have the author's name on the top.
5. All source code must be written in Java and commented reasonably well.
6. You are not allowed to use library-provided subroutines if they are what you are asked to implement.
7. Sharing any content of this assignment and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Topological Sorting of a DAG Using Graph's DFS

In this programming assignment, you will implement the topological sorting of a directed acyclic graph (DAG) using the graph's depth first search (DFS). Please refer to the lecture slides and book chapter for the algorithm that solves this problem.

Specification

- The Java class that has the `main` function needs to be named as "TopoSort". That is, the name of the source file that contains the `main` function should be "TopoSort.java".
- **The input of your program** is a text file that represents a given DAG in its list representation. For example, suppose the following is the DAG, for which you need to do the topological sorting:



then, the given file, whose name shall be supplied by the grader as a command line parameter, may contain the following content:

```
0:1,2,3
1:3
2:
3:2
```

In particular, the input file has multiple lines, where each line has two parts separated by a colon: the first part represents a graph vertex; the second part is the collection of the vertex's next hop neighbors separated by commas.

You are guaranteed that all the vertices in the graph are named as $0, 1, \ldots, n-1$, where n is the number of vertices in the graph. However, remind that a given graph can also be represented by other different input text files. For example, the same example graph above can also be represented by the following two text files.

```
1:3
0:3,1,2
2:
3:2
```

```
1:3
0:2,1,3
3:2
2:
```

In other words, we do not have any particular requirement on the order of the vertices that are listed before the colon. We also do not have any particular requirement on the order of the next-hop neighbors in a particular line of the input file. **Your program needs to be able to work with any one of these possible input files, all of which are representing the same graph.**

Note: (1) The given graph is guaranteed to be a DAG. (2) The number of nodes in the graph can be as small as one.

- **The output of your program** is the topological sorting of the given DAG, which is simply a sequential print of all the graph vertices, where each vertex will be printed exactly once, such that for any pair of two vertices u and v in the graph, if there is at least one route going from u to v (meaning there is no route going from v to u, because the given graph is a DAG), then u must be printed before v.

  For example, suppose the grader provides the aforementioned example DAG as an input by providing its filename as `dag.txt`, the command line to run your program will be:

  ```
  $java TopoSort dag.txt
  ```
  (Note: `$` is the command line prompt and is not part of the command line.)

  Your program's output, which is the topological sorting of the given DAG, will be:

  `0, 1, 3, 2`

  Note that, in general, you may have multiple topological sorting of a given DAG, your program only needs to find and print one of them.

A few suggestions.

- Make a good plan and design for your program and decompose it into a few loosely coupled components. Do not work on the next component before the previous one is finished and well tested. You will not succeed in debugging a large program, if it is built upon a collection of buggy components.
- Finish a robust component that handles the reading of the input file.
- For a speedy code production, you are allowed to Java-provided data structures for the linked list, for example, the array list.
- Finish a robust component that handles the loading/construction of the graph's list representation in the RAM.
- The array in the list representation should be an array of linked list class data type, rather than an array of linked list node class data type. The later choice is a bad design.
- Finish a robust implementation of a graph's DFS (both the `DFS(G)` and `graph_DFS(G,s)` functions). This DFS implementation should be working for any graph, not only for a DAG.
- Design and decide where and how to save various tracking information within each vertex, in order to make the DFS serve the topological sorting of the given DAG.
- Consider all possibilities in your implementation to make your program/product robust.

Submission

- All your work files must be saved in one folder, named: **firstname_lastname_EWUID_cscd320_prog4**
  (1) We use the underline '_' not the dash '-'.
  (2) All letters are in the lower case including your name's initial letters.
  (3) If you have middle name(s), you don't have to put them into the submission's filename.
  (4) If your name contains the dash symbol '-', you can keep them.
  (5) You do NOT need enclose the testing data files and the Java .class files.
- You then compress the above whole folder into a .zip file.
- Submit .zip file onto the Canvas system by the deadline.