CPE 349: Assignment 2– Topological Sorting

Your goal is to implement two algorithms that take as input a directed graph and produces a topological sort of the vertices if the graph is a DAG. If the graph is not a DAG the program should report this. (See below for details.) Your algorithm must have a worst case performance of O(|E|+|V|)

Deliverable: A class called **TopSorter.java that** contains

1. A static method that tries to find a topological sort of the vertices of the graph using depth first search. The class must have a method **dfsTopSorter**. **dfsTopSorter** has a single input parameter (String type) that is the filename containing the graph to be topologically sorted. If the graph is a DAG it will return an ArrayList of Integers of a valid topological sort from the graph. The signature of your method must be: **public static ArrayList<Integer> dfsTopSorter** (String)
2. A method that tries to find a topological sort of the vertices of the graph using the source removal algorithm discussed as part of this class. The class must have a method **srcRemTopSorter**. **srcRemTopSorter** has a single input parameter (String type) that is the filename containing the graph to be topologically sorted. If the graph is a DAG it will return an ArrayList of Integers of a valid topological sort from the graph. The signature of your method must be:
   public static ArrayList<Integer> srcRemTopSorter (String)

Modify the **GraphStart3 class provided as a base for** your **TopSorter** class and then adding **dfsTopSorter and srcRemTopSorter** methods and any necessary helper functions. Submit the **TopSorter,java** class to Canvas.

Requirements and constraints

1. Here the graph is directed and an input file consists of one test case.
   - Each test case will begin with a line containing 1 or 0 to tell your program if the graph is directed or undirected. **For this assignment the line will always contain 1.**
   - The second line contains an integer **n**, that is the number of vertices in the graph to be tested, where 1<n<200. **Each vertex in the graph is represented by a number from 1 to n.**
   - The third line is the number of edges, **e**
   - This is followed by **e** lines each containing a pair of integers (each integer is between 1 and n) that represents an edge between the vertices represented by the numbers. The edge "goes" from the first vertex to the second vertex in the pair.
2. The method **dfsTopSorter and srcRemTopSorter** methods must return, if possible, a valid topological sort in an ArrayList. For example, the array list for 5 vertices might be 3, 2, 4, 1, 5 where 3 comes first in the proposed linear order. If there is no topological sort for the graph (not a DAG) then the ArrayList should contain -1's.
3. **Your Source Removal Algorithm may NOT use recursion** and must be based on the **algorithm** covered in the screencast. The DFS based algorithm **must be recursive**.
4. Clearly comment your code.
5. We will call your class and method assuming the signatures specified.
   ArrayList<Integer> list = **TopSorter.srcRemTopSorter** ("filename")
   ArrayList<Integer> list = **TopSorter.dfsTopSorter** ("filename")

Example: **Input parameter is a string that is the name of a text file containing:**

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

Might return: `1 2 3 4 5 // or any another valid topological sort`

Example: **Input parameter is a string that is the name of a text file containing:**

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

Returns: `-1 -1 -1 -1 -1`

Avoid the usual pitfalls:

- Use the exact Class name specified (check spelling!)
- Use the exact method signatures specified
- Remove all package statements and extraneous imports, remove all print statements