CPE 349: Assignment 3– Topological Sort using Source Removal

Your goal is to implement an algorithm that takes as input a directed graph and produces a topological sort of the vertices if the graph is a DAG and otherwise produces a partial topological sort of the vertices of the graph. (See below for details.)
Your algorithm must have a worst case performance of O(|E|+|V|)
Deliverable: Two .java files

1. A class containing a method that tries to find a topological sort of the vertices of the graph. The class must be called **TopSorter** and have a method **topSortGenerator**. **topSortGenerator** has a single input parameter (String type) that is the filename containing the graph to be topologically sorted. It will return an ArrayList of Integers of a valid topological sort from the graph.
   public ArrayList<Integer> topSortGenerator (String)
2. Use the **GraphStart class provide earlier** in your TopSorter class. **Both** classes will be submitted.

Requirements and constraints

1. Here the graph is directed and the 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 **topSortGenerator** must return the proposed 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 the vertices that can be topologically sorted followed by -1's for the missing vertices. If no partial sort exists then return an ArrayList containing n -1's (recall n is the number of vertices)
3. Your algorithm may NOT use recursion and **must be based on the Source Removal Algorithm** covered in class.
4. Clearly comment your code.
5. We will call your class and method assuming the above signature with the call:
   ArrayList<Integer> list = TopSorter.topSortGenerator("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
```

Returns:

```
1 2 3 4 5
```

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