Homework Assignment #10: Programming Exercise #2

Due Time/Date

2:20PM Tuesday, December 12, 2023. Late submission will be penalized by 20% for each working day overdue.

Problem Description

Implement the algorithm (discussed in class) for computing the strongly connected components of a directed graph. (Note: you may want to take this opportunity to try the two different ways of updating the *High* value of a vertex when it sees another "undetermined" vertex through a cross or back edge.)

Please follow the input format as described below. The first line of an input contains one integer n ($\leq$ 1000), indicating the number of vertices in the graph; the vertices are then identified by numbers 1 through n. Each of the following lines represents the adjacency list of a particular vertex u, with the first integer giving the identifier of u followed by the identifiers of those vertices (in no particular order) that are connected by an edge from u. Below is a sample input file:

```
9
1 2 8
2 3
3 1 4 7
4 5
5 6
6 4
7 5
8 3 9
9 7
```

For the output, each line contains the identifiers of a strongly connected component.

```
4 5 6
7
9
1 2 3 8
```

Important Notes

This assignment constitutes 4% of your grade. You may discuss the problem with others, but copying code is strictly forbidden. **Some of you may be requested to demonstrate your program.**

Submission Guidelines

- Pack everything, excluding compiler-generated files, in a .zip file, named with the pattern "`b117050xx-alg2023-hw10.zip`".
- Upload the .zip file to the NTU COOL site for Algorithms 2023.
- If you use a Makefile, make sure that it outputs "`hw10`". Otherwise, make sure that the whole application can be compiled by a single command like "`gcc hw10.c`", "`g++ hw10.cpp`", 'javac `hw10.java`", etc.

Grading

Your work will be graded according to its correctness and presentation. Before submission, you should have tested your program on several input cases. You should organize and document your program (preferably as comments in the source code) in such a way that other programmers, for example your classmates, can understand it. In the documentation of your program, you are encouraged to describe how you have applied the algorithmic techniques, in particular design by induction and/or reduction, learned in class.

Below is a more specific grading policy:

Criteria	Score
incomplete or doesn't compile	≤ 20
complete, compiles, but with major errors	≤ 40
complete, compiles, but with minor errors	≤ 70
correct (passing all test cases)	≤ 90
correct and reasonably efficient (at least around the class-average)	≤ 100
well-organized and with helpful code comments	$+ \leq 10$