Algorithms
Homework 2 Specification

Strongly Connected Components

Input format. The first line contains two integers: n (number of vertices, $n \leq 10^4$) and m (number of directed edges, $m \leq 10^6$). Each of the following m lines contains two integers u and v, denoting a directed edge $u \to v$. Vertices are labeled $0, 1, \ldots, n-1$.
e.g.

```
10 15
0 1
0 2
0 3
1 4
4 5
5 6
6 2
6 7
2 3
2 4
3 7
7 8
8 9
9 7
3 6
```

Output format. Print each strongly connected component (SCC) on its own line as a space-separated list of vertex labels. Within each line, vertices must be sorted in increasing order, and the list of lines (i.e., the SCCs) must be in lexicographic order.

On the last line, print the time (in milliseconds) taken **solely to compute SCCs using DFS.** This should **exclude time spent on input parsing, building the transpose graph, and sorting the output SCCs.** Only the core algorithm (two DFS passes) should be measured.
e.g.

```
0
1
2 3 4 5 6
7 8 9
0.029833
```

Implementation. You are required to implement a program that finds all SCCs from a directed graph input, and prints the output in the format specified above. Although the input graph is provided in only one format (edge array), you are required to convert it into all three representations (adjacency matrix/list/array) and implement a separate program or code for finding SCCs using each representation. **Unlike Homework 1, no skeleton code will be provided.** Your program must correctly handle all three types of graph formats.

Compilation and Execution. Clearly specify the commands required to run your program for each of the three graph formats. We recommend following the compilation and execution style used in Homework 1 to ensure compatibility. Please include these instructions in a `readme.txt` file. An example is shown below:

```
# Compile
gcc -O2 -std=gnu99 -Iinclude src/main.c src/scc.c -o main

# Run with adjacency matrix graph format
./main -m < example.txt > output

# Run with adjacency list graph format
./main -l < example.txt > output

# Run with adjacency array graph format
./main -a < example.txt > output
```

Datasets. Additionally, real-world datasets will be provided. Table 1 summarizes the number of vertices, number of edges, and edge density ($\frac{|E|}{|V|(|V|-1)}$) for each dataset.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	**Density**
congress-Twitter	475	13,289	0.0591
email-Eu-core	986	24,929	0.0257
soc-bitcoinalpha	3,783	24,186	0.0017
p2p-Gnutella08	6,301	20,777	0.0005

Table 1: Statistics for various real-world directed graphs

Grading

For each instance, your program may use up to 1 minute for C/C++ (3 minutes for Python and Java) and 1 GB of memory. **You are not allowed to use any non-standard libraries or built-in libraries that provide graph algorithms, sorting or similar functionality. If you are unsure whether a library is allowed, please contact the TA.**

Environment. We suggest using one of C, C++, Java or Python. If it is difficult for you to use one of the languages mentioned above, please contact the TA (`ta@theory.snu.ac.kr`). The compilers/interpreters that will be used for grading are as follows:

- C: gcc (Debian 12.2.0-14) 12.2.0
- C++: g++ (Debian 12.2.0-14) 12.2.0
- Python: Python 3.11.5
- Java: javac (OpenJDK-23)

Submission. Please compress your submission as `AG_HW2_[student ID].zip` (e.g., `AG_HW2_2025-12345.zip`). Your submission should include at least 4 files:

- Source code for the problem
- Report (`report.pdf`)
- One additional example (`example.txt`)
- A description of how to run your program for all three graph formats (`readme.txt`)

Hand in your submission to eTL. Please make sure that the submission files are placed at the top level of the compression and that submission file names are the same as above. **Please follow the rules for submission file names.**