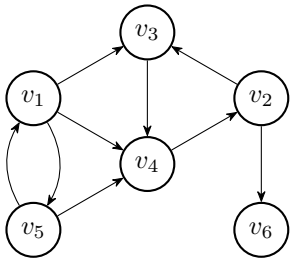
Assignment 3 — Directed Graphs

In a directed graph, the existence of a path from a vertex u to a vertex v does not imply the existence of a path from v back to u. As a result, finding a path from u to v does not necessarily mean that they are in the same component. Finding the components of a directed graph requires a more sophisticated approach that a pure depth first search.

1 Strongly Connected Components

Recall that a *strongly connected component* of a directed graph is a maximal set of vertices within which there exists a path between every pair of vertices. For example, consider the following directed graph:



This graph contains three strongly connected components: $\{v_1, v_5\}$, $\{v_2, v_3, v_4\}$, and $\{v_6\}$. Notice that v_1 alone does not form a component, because it's not *maximal*; v_5 can still be added without breaking its strong connectivity.

In your programming language of choice, implement an algorithm to find the strongly connected components of a graph. To help you get started, not the following observations:

- No strongly connected component can span multiple trees in a forest generated by a depth-first search.
- Assigning pre- and post-visit numbers to vertices during a depth-first search allows identification of back edges.
- Finding a back edge during a depth-first search indicates the existence of a cycle.
- All vertices along a cycle must be in the same strongly connected component.
- If two cycles share any vertices, then they are both in the same strongly connected component.

Each input graph will be provided as an *edge list*: each edge in the graph will be represented by a space-separated pair of vertex identifiers, indicating an edge from the first vertex to the second.

You may assume that vertex identifiers are contiguous positive integers starting from 1 (i.e. they begin at 1 and contain no "gaps"). You may also assume that the graph will be simple and will not contain any isolated vertices.

For example, the above graph could be represented as:

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

Your program must accept as a command line argument the name of a file containing an edge list as described above, the print to `stdout` the strongly connected components as follows.

- Print the number of strongly connected components.
- Print each strongly connected component as a comma separated list on its own line.

For example:

```
$ ./compile.sh
$ ./run.sh test_files/in1.txt
3 Strongly Connected Component(s):
1, 5
2, 3, 4
6
```

The correct answer is unique up to ordering. Your answer may list the components in a different order and may also list the vertices within a component in a different order; that's fine. Taking the time to sort the output would worsen the efficiency of the algorithm.

2 Submission

The following items must be demonstrated/presented to me in lab on the day of the deadline.

- Pseudocode for an efficient algorithm to find the strongly connected components of a directed graph.
- Drawings of three graphs that you consider interesting test cases for your algorithm.

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `compile.sh` — A bash script to compile your submission (even if it does nothing).
- `run.sh` — A bash script to run your submission.
- `*.py` or `*.java` — Your source code in Python or Java of a working program to find strongly connected components.