Assignment 3 — Undirected Graphs

Due: Monday, October 14th

Graphs allow us to mathematically and computationally model and explore the relationships that occur in real world situations. One possible relationship that we might be interested in is some form of conflict between two entities. For example, vertices could represent university classes, where two vertices are connected by an edge if those classes' times overlap.

Deliverables:

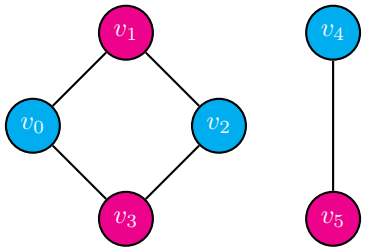
GitHub Classroom: `https://classroom.github.com/a/TbnVCgQo`

Required Files: `compile.sh`, `run.sh`

Optional Files: `*.py`, `*.java`, `*.clj`, `*.kt`, `*.js`, `*.sh`

Part 1: Graph Colorings

A *vertex coloring* is an assignment of colors, one color to each vertex, such that no two adjacent vertices have the same color. If a graph can be colored using k colors, it is said to be *k-colorable*. In the situation described above, where edges connect classes with conflicting times, a coloring could represent classroom assignments such that no two classes are in the same classroom at the same time. For example:



The above graph can be colored using two colors, and it is said to be 2-colorable. For this special case of $k = 2$, the 2-colorability of a graph can be computed in linear time (for $k \geq 3$, finding valid colorings generally becomes very difficult).

In your programming language of choice per Assignment 1, implement an algorithm to determine whether or not a given graph is 2-colorable. To help you get started, note the following observations:

- In order to establish the 2-colorability of a graph, every vertex must be assigned exactly one color.
- Thus, every vertex should be explored exactly once by the coloring algorithm.
- The choice of cyan and magenta in the example above was entirely arbitrary; any two colors would do.
- Moreover, the choice of *which* group of vertices was colored cyan was also arbitrary: the assigned colors could be swapped, and the coloring would remain valid.
- The colors assigned within one component cannot affect the colors assigned within any other component. However, if any one component is not 2-colorable, the entire graph is not 2-colorable.

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

You may assume that vertex identifiers are contiguous natural numbers — they begin at 0, and there will be no "gaps" in the identifiers used. 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:

```
0, 1
1, 2
2, 3
3, 0
4, 5
```

Your program must accept as a command line argument the name of a file containing an edge list as described above, then print to `stdout` the result of the attempted 2-coloring according to the following format:

- Vertices assigned the same color within the same component appear on a single comma-separated line.
- Vertices assigned different colors appear on different lines. Vertices in different components appear on different lines, even if they are assigned the same color.
- Each line's vertices must be sorted in ascending order. Note that vertex identifiers are integers, not strings. The lines themselves must be sorted in ascending order using their smallest vertex identifiers.

For example:

```
>$ ./compile.sh
>$ ./run.sh in1.txt
Is 2-colorable:
0, 2
1, 3
4
5
```

Your program will be tested using `diff`, so its printed output must match *exactly*.

Part 2: Submission

The following items must be demonstrated in lab on the day of the deadline:

- Pseudocode for an efficient algorithm to 2-color simple graphs.
- Analysis of complexity for the pseudocode.

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), as specified.
- `run.sh` — A Bash script to run your submission, as specified.

The following files are optional:

- `*.py`, `*.java`, `*.clj`, `*.kt`, `*.js`, `*.sh` — The Python, Java, Clojure, Kotlin, Node.js, or Bash source code files of a working program to 2-color simple graphs, as specified.

Any files other than these will be ignored.

Additionally:

- On the date before the due date, the grader will be run after 8pm and provides feedback containing only what your program scores. Only submissions made before 8pm are guaranteed to receive feedback. What your program scores on the official run (the due date) is the final score.
- Late submissions made within 24 hours of the deadline receive a .7 multiplier. If you decide to make a late submission, please notify me by sending an email to vnguy143@calpoly.eduwith both the subject and the body in the format: "CSC349,late,asgn<i>", with i being the assignment number.