Reminder: as specified in the Computer Science Academic Integrity Policy, you should not **look at** any other student's code. You are free to discuss data structures, general approaches, and so on; for any specific debugging help you should come to me.

On the course website you will find several data files describing directed graphs. The format of each file is as follows:

- The first line contains a single integer n giving the number of nodes in the graph.
- The second line contains a single integer m giving the number of edges.
- Each of the following m lines contains a pair of integers x and y separated by a space, indicating a directed edge from node x to node y. The nodes are numbered 0 through $n - 1$.

  For example, `demo-cycle.in` looks like this:

```
10
18
5 0
1 8
2 5
3 5
4 0
6 2
9 3
4 9
6 7
7 4
7 5
9 5
2 9
4 1
7 0
2 7
6 3
4 6
```

It describes a directed graph with 10 nodes (numbered 0 through 9) and 18 edges. The edges are $(5, 0)$, $(1, 8)$, and so on.

For each input graph, you should determine whether it is a DAG or not (that is, whether or not it is acyclic). In addition:

- If it is a DAG, you should specify a topological ordering.
- If it is not a DAG, you should specify a directed cycle in the graph.

For each input graph, you should create an output file with the following format:

- For `<FILENAME>.in` the corresponding output file should be called `<FILENAME>.out`.

- The first line of the output file should contain either the string `DAG` or the string `cycle`, indicating whether or not the input graph is a DAG.
- If the first line contains `DAG`, the remaining lines should list all the vertices of the graph in some topological ordering, with one vertex per line.
- If the first line contains `cycle`, the remaining lines should list the vertices contained in some directed cycle. The first vertex should *not* be repeated at the end.

For example, one correct output for `demo-cycle.in` is contained in `demo-cycle.out`, and looks like this:

```
cycle
6
7
4
```

As you can verify, the graph described in `demo-cycle.in` does indeed contain the edges $(6,7)$, $(7,4)$, and $(4,6)$, which form a directed cycle.

Likewise, a correct output for `demo-dag.in` (which is a DAG) is contained in `demo-dag.out`. Note that in general there may be many different correct output files, since a graph may contain many different cycles or many possible topological orderings.

There are four sizes of graph: `tiny` ($|V| = 10$), `small` ($|V| = 10^3$), `med` ($|V| = 10^5$) and `big` ($|V| = 10^6$). The graphs are *sparse* in the sense that they have only $O(V)$ edges. Note that a $\Theta(V^2)$ algorithm will probably work for `tiny` and `small` (and *perhaps* even for `med` if you are willing to wait long enough), but you will almost certainly need a $\Theta(V+E)$ algorithm (which is $\Theta(V)$ for these graphs) to solve the `big` graphs.

For each size, there are two graphs. You should **not** assume that there is one DAG and one cyclic graph of each size!

You can use the provided Python program `check-graph.py` to check your answers. Simply provide it with the name of the input graph file and the name of your output file, like this:

```
$ python check-graph.py demo-cycle.in demo-cycle.out
Checking cycle...
Cycle looks good!
```

If the output is incorrect, the tool will give a counterexample, for example:

```
$ python check-graph.py demo-cycle.in demo-dag.out
Checking DAG...
Found bad edge: 2 5
```

This means that although the output file claims to give a topological ordering for the DAG, the edge $(2,5)$ is contained in the graph and goes backwards from a later node in the ordering to an earlier one, so it is not in fact a topological ordering.

What to turn in

- Create a `.zip` file containing:
  - Your output files. Remember that the output file for `<FILENAME>.in` should be named `<FILENAME>.out`.
  - Any program(s) you used to generate the output files.
- Visit `https://goo.gl/forms/VxtMnNXTVe5qOOD73` to submit your `.zip` file. (If you open this document as a PDF you can simply click the URL above.) If you do not have a Google account and do not want to make one, you may email me the `.zip` file instead.

Grading scheme Each input graph is worth a certain number of points as follows:

- `tiny`: 2 points each
- `small`: 3 points each
- `med`: 3 points each
- `big`: 2 points each

giving a total of 20 possible points. Full credit will be given for any correct output file. **Half credit will be given for an output file which correctly specifies `DAG` or `cycle` but does not give correct evidence for it.** (For example, if you determine that an input graph must have cycles but you do not know how to find one, you could just turn in an output file containing the single line `cycle`, with no list of vertices following; this would earn half credit.)

If you know a graph is a DAG or has a cycle, but aren't able to generate evidence for it, *at the very least* turn in an output file that just says `DAG` or `cycle`!

You are not *required* to write a program, but you will probably not be able to get many points otherwise (you can certainly construct correct `tiny` outputs by hand—in fact, I recommend it, as a way to determine whether your program is behaving correctly!—but even the `small` graphs, with $n = 10^3$, would probably be too tedious to do by hand). You **must** turn in any programs you write, though the programs themselves will not be graded.