CMPSC 250
Analysis of Algorithms
Fall 2012
Bob Roos

Lab 7
Tues., 6 November 2012
Hand In by 2:45pm Tues., 13 Nov.

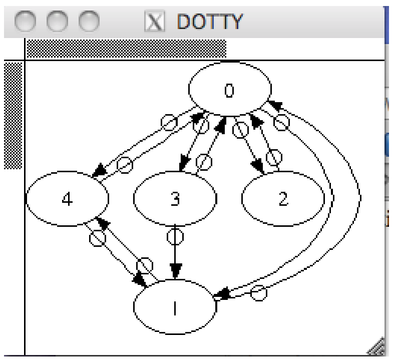
From now on I will be deducting points from programs that do not contain header comments for each method you write (including their purpose, their return value if any, and the meanings of the parameters) and descriptive comments for the program as a whole (including comments on what it does and how to use it, e.g., command line arguments and their meanings), descriptions of any variables that you declare, etc. I will also be deducting points for any documents, including programs, that fail to contain your name and the lab number.

The `Digraph` Class

Make sure you know how to create Digraphs either by reading from a file or by explicitly adding edges. The process is the same as for the `Graph` class.

For instance, here is how to create a directed graph in which there is an edge from i to j if either $i > 2j$ or $j > 3i$; the number of vertices is given in the first command line argument, `args[0]` (we used 5 to obtain the graph on the right):

```
  ...
int numV = Integer.parseInt(args[0]);
Digraph g = new Digraph(numV);
for (int i = 0; i < numV; i++) {
  for (int j = 0; j < numV; j++) {
    if (i==j)  // don't allow loops
       continue;
    if (i > 2*j || j > 3*i)
       g.addEdge(i,j);
  }
}
  ...
```

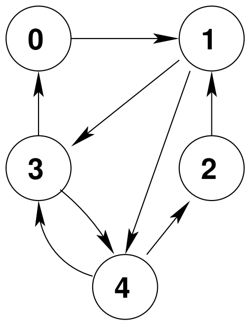


1. **Notation:** If a and n are positive integers, $a(\text{mod } n)$ is equal to the remainder when a is divided by n. In Java, this is equivalent to writing `a % n`.

Write a Java program that creates a digraph of V vertices, where V is given on the command line. There should be an edge from v to w if either of the following conditions holds:

- $v = (2w)(\text{mod } V)$
- $(w^2)(\text{mod } V) = (3v + 1)(\text{mod } V)$

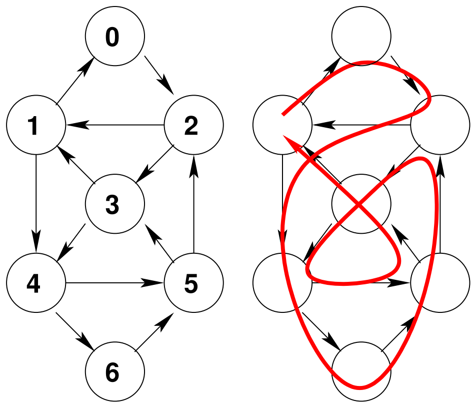
For example, if $V = 5$, there is an edge $0 \to 1$ because $1^2 = (3 \cdot 0 + 1)(\text{mod } 5)$; there is an edge $1 \to 3$ because $1 = (2 \cdot 3)(\text{mod } 5)$. The full graph for $V = 5$ looks like:



Your program should print out the graph description in adjacency list format.

To hand in: Submit your program and outputs for the cases $V = 3$, $V = 4$, $V = 5$, and $V = 6$.

2. A digraph is said to be *Eulerian* if it is possible to construct a path, possibly with repeated vertices, that uses every edge *exactly once*, returning to the starting point. For instance, the graph below is Eulerian as shown by the path in red on the right:



Eulerian cycle: 1, 0, 2, 1, 4, 6, 5, 2, 3, 4, 5, 3, 1

There is a very simple test to see if a digraph is Eulerian: **for each vertex v, the in-degree must equal the out-degree.**

Write a Java program that inputs a graph from a file (I will provide you with some test files); the file name should be specified on the command line. By traversing the adjacency lists of the vertices, determine the in-degree and out-degree of every vertex and then print out whether or not the graph is Eulerian. Sample `main` program:

```
  ...
public static void main(String[] args) {
    Digraph g = new Digraph(new In(args[0]));
    if (euler(g))
```

```
            StdOut.println("eulerian");
        else
            StdOut.println("not eulerian");
    }

    public static boolean euler(Digraph g) {
      ...
     return true if g is Eulerian, false otherwise
      ...
    }
```

HINT: You have already seen how to compute the in-degrees of the vertices in program `NaiveTopSort.java` on Nov. 5.

To hand in: Submit your program and outputs from the four test files I've provided.

3. [**Extra credit.**] Determine by hand Euler cycles for the test files that contain Eulerian digraphs—write down the sequence of nodes visited.

4. [**Extra credit.**] Write a Java program that inputs two digraphs G_1 and G_2 and determines whether or not they are equal. We say that they are equal if, for each vertex v, the adjacency list for v in graph G_1 contains exactly the same vertices as the adjacency list for v in graph G_2, but possibly in a different order. Create digraph files to test it and show the files and your results.

5. [**Extra credit.**] If G is a digraph, its *underlying graph* is the undirected graph obtained by changing directed edges to undirected edges and then removing duplicate edges. For example, if a digraph has an edge $v \to w$ we will have an edge between v and w in the underlying undirected graph. However, if we the digraph *also* has an edge $w \to v$, we will not add a second edge between v and w in the underlying graph.

   Write a Java program that inputs a digraph from a file, constructs the corresponding underlying graph (without multiple edges between vertices), and prints it out in adjacency-list format. Test it and show the test files and your results.

6. [**Extra extra credit if you did the previous one.**] A digraph is called *weakly connected* if its underlying graph is connected. Write a Java program that inputs a digraph from a file and prints whether or not it is weakly connected. (Use previous answer plus depth-first search.)

7. Move up one level from your `lab7` directory and type "`zip -r lastnamelab7.zip lab7`" (if you named it something else, use the something else). Now upload this to your drop box.