CS 61B Lab 13
April 24-April 25, 2014

Goal: to give you experience with an unweighted directed graph represented by an adjacency matrix.

Please make sure you have a partner for this lab.

Copy the Lab 13 directory by doing the following, starting from your home directory.

```
cp -r ~cs61b/lab/lab13 .
```

The file UDGraph.java contains code for a class UDGraph, an unweighted directed graph represented by a boolean adjacency matrix adjMatrix[][]. For simplicity, vertices are denoted by ints in the range 0...n - 1, where n is the number of vertices. Each entry adjMatrix[i][j] is true if (i, j) is an edge of the graph; false otherwise.

The UDGraph class currently has the following methods.

```
UDGraph(int n)  Constructs a new UDGraph with n vertices and no edges.
getNumVertices()  Returns the number of vertices in the UDGraph.
getNumEdges()  Returns the number of edges in the UDGraph.
validVertex(int v)  True if v is a valid vertex number (0...n - 1).
hasEdge(int o, int d)  True if the graph contains edge (o, d).
addEdge(int o, int d)  Adds edge (o, d) to the graph (if not already there).
removeEdge(int o, int d)  Removes edge (o, d) from the graph (if there).
toString()  Returns a String representation of the UDGraph.
```

You are welcome to use these methods and/or manipulate the fields "adjMatrix", "vertices", and "edges" directly, as you prefer. The addEdge and removeEdge methods have the advantage that they update the "edges" count correctly (always checking whether the edge is present in the graph before the update).

Part I: Finding vertices reachable by length-2 paths (2 points)

Fill in the body of the method UDGraph.length2Paths(). This method constructs and returns a UDGraph with the same number of vertices as "this" UDGraph. The new graph contains the edge (v, w) if and only if there is a path of length 2 from v to w in "this" graph--in other words, there is some vertex u such that (v, u) and (u, w) are both edges of "this" graph.

Note that a length-2 path can start and end at the same vertex: if "this" graph contains the edges (v, w) and (w, v), then it contains a length-2 path <v, w, v> from v to itself, and the new graph should contain the self-edge (v, v). Moreover, if "this" graph contains the self-edge (v, v), then <v, v, v> is a length-2 path, so the new graph should contain (v, v).

If a vertex w can be reached from a vertex v by a length-1 path (one edge) in "this" graph but _not_ by a length-2 path, the new graph should _not_ contain (v, w).

Try to think of the fastest, simplest code for length2Paths(). It's possible to do it with a relatively simple triply-nested loop. You will have to explain your algorithm to your TA. If your TA thinks your algorithm is too slow, you'll be asked to do it again.

Your solution should not change "this" graph.

Part II: Finding vertices reachable by length-k paths (2 points)

Fill in the body of the method UDGraph.paths(int length). This method creates and returns a UDGraph with the same number of vertices as "this" UDGraph. The new graph contains the edge (v, w) if and only if there is a path of length "length" (the parameter) from v to w in "this" graph. Your method should work for any "length" of 2 or greater.

Note that a length-k path is permitted to use an edge multiple times. For example, <u, v, w, u, v, w> is a valid length-5 path.

Hint: First calculate all the paths of length (k - 1). Once you know these, it's straightforward to compute all the paths of length k in a manner similar to what you did for Part I.

There is test code in UDGraph.main() for both Parts I and II.

Check-off

```
2 points:  Show your TA your code for length2Paths() and explain how you did
           it.  Run the test code to show that length2Paths() works.
2 points:  Run the test code to show that paths() works.
```