Project 2 **Scott Barlowe**
CS 351, Fall 2016

O Vertex, Where Art Thou?

OVERVIEW

In this assignment, you will use the iterative algorithms discussed in class to construct a program for performing depth-first searches (DFS), transitive closure (TC), and cycle searches in directed graphs. Specifically, your program should

Minimum Requirements

1. Read a file of input values input as a command line argument.
2. Store the input graph as a list of adjacent vertices for each Vertex object.
3. Prompt the user for a source vertex and destination vertex (in that order) represented by integers.
4. Perform error checking to make sure the file can be read and that the source and destination are in the graph.
5. Perform a DFS to find the destination from the source specified by user input.
6. Find the transitive closure of the graph.
7. Check the graph for cycles.
8. Print the following information
   - DFS → If the destination was found or not found. If found, prints 1) the vertices in the order in which they were discovered and 2) the first discovered path to the destination. If not found, print "Not Found".
   - TC → Outputs the edges that have been added in the same format as the input (see below).
   - Cycle → If a cycle exists or not (on the original graph).

PROGRAM STRUCTURE

Your program should consist of a **GraphDriver** class, a **Graph** class, and a **Vertex** class. Additionally, your program should follow the structure below but you can change the return/data types and parameters.

The **Vertex** class should have at least the following fields. You may add any fields and methods that you wish.

```
Integer ID          // Unique Integer corresponding to a vertex's id number. Vertices are
                    // numbered from 0 and range to (number of vertices – 1).

String color        // Color of the vertex
```

The **Graph** class should have at least the following fields and methods. You may add any additional fields and methods that you wish.

Fields for the Graph class:

```
ArrayList<Vertex> vertexList            // List of vertices in the graph where the index in vertexList corresponds
                                           to ID.

ArrayList<ArrayList<Integer>> adjList   // Adjacency list where the list at each index corresponds to the vertices
                                           adjacent to vertex ID in vertexList

int [ ] [ ] adjMatrix;                  // Adjacency matrix where a value of 1 represents an edge between
                                           two vertices.
```

Methods for the Graph class:

```
**************************************************************************
Initiates and runs the entire process described 1 – 5
@params:  String graphFile -- Name of the input file to be read for the adjacency list
**************************************************************************/
```

void StartGraph(String graphFile)

```
/**************************************************************************
Reads the inputFile given at a command line argument and places its contents
into a master list vertexLis and adjacency list adjList.
@params:  String graphFile -- Name of the input file to be read
**************************************************************************/
```

readInputGraph(String inputFile)

```
/**************************************************************************
Prompts, checks, and accepts valid arguments for the source and destination
**************************************************************************/
```

readSourceDest( )

```
/**************************************************************************
Performs the depth-first search form source to destination or until the search is exhausted
**************************************************************************/
```

dfsSearch

```
/**************************************************************************
Performs the search for cycles
**************************************************************************/
```

cycleSearch

```
/**************************************************************************
Performs the transitive closure
**************************************************************************/
```

transitiveClosure

```
/**************************************************************************
Prints the information as specified as above (and below).
**************************************************************************/
```

printGraphStats

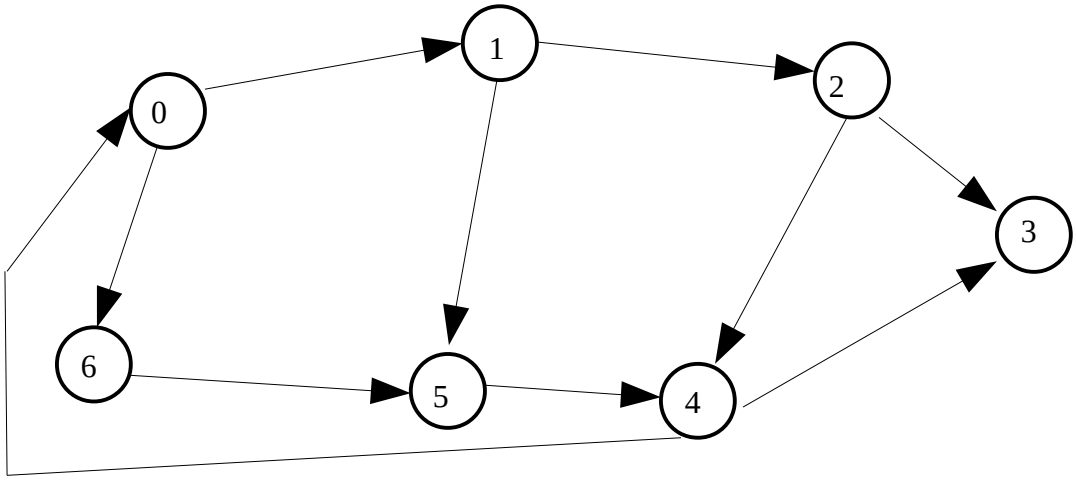
The **GraphDriver** class should only create a **Graph** object and call its **StartGraph** method.

INPUT

Your program should take the names of one input file as a command line parameter. Each line in the command line argument has two entries that represents an edge in the graph. For example the input

0 1
0 6
1 2
1 5
2 3
2 4
4 3
4 0
5 4
6 5

corresponds to



The user is prompted for a source-destination pair. For example the input

0 3

would represent a search from vertex 0 to vertex 3. When deciding which adjacent vertex to process next (all other things being the same), choose the adjacent vertex with the lowest ID. For example, the output for the graph above would be

[DFS Discovered Vertices: 0, 3] *Vertex A, Vertex B, ….*
[DFS Path: 0, 3] *Vertex A → Vertex B → ….*
[TC: New Edges] 4 1
4 2
…....
[Cycle]: Cycle Exists

You may complete the assignment in any language you wish. If you complete the project in a language other than Java, you must supply a README file describing how to compile and/or execute your program and follow the structure above as closely as possible. You may talk to other teams about the concepts in the project but you are not allowed to share any code.

All programs must compile/execute on agora. Your program (if completed in Java) should compile and execute on agora with

>>>javac *.java

>>>java GraphDriver input.txt

(continued on next page)

The assignment is due by midnight on November 2. You may turn in your project late for a penalty of 5 points per day. After five days, projects will not be accepted. Additionally, your program may be tested with multiple input graphs. Submit your files to agora as a jar or tar file using the following command:

>>>handin.351.1 2 ***myfile***