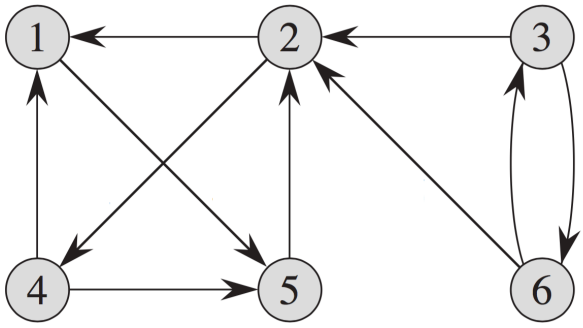
NETS 150 – Homework 1

Due – Feb 7, 2019 at 12.00pm

Part 1 – Theory (20 points)

Please do the following problems:

1. Consider the graph shown below:



   a. What is the in-degree and out-degree of each node? (2 points)
   b. Show how Breadth-First Search works on the graph starting at node 2 (i.e., show the order of nodes visited). (4 points)
   c. Show how Depth-First Search works on the graph starting at node 2 (i.e., show the order of nodes visited). (4 points)

2. There are two kinds of professional actors: "Mainstream" actors (i.e., actors who will generally be a part of big-budget studio movies) and "Indie" actors (i.e., actors who will generally be a part of smaller, independent movies). Between any pair of actors, there may or may not be a rivalry. Suppose we have *k* actors and we have a list of r pairs of actors for which there are rivalries. Give an algorithm that determines whether it is possible to designate some of the actors as mainstream or indie such that each rivalry is between a mainstream actor and an indie actor. The algorithm should run in O(k+r) time. If it is possible to perform such a designation, your algorithm should produce it. (10 points)

Part 2 – Programming (80 points)

Facebook friends: For the programming part of this homework, we will use real-world data from Facebook. Please see the file attached: `facebook_combined.txt`. This data contains 4039 nodes and 88234 edges. The nodes correspond to people on Facebook and the edges to friendships. As friendships in Facebook are bi-directional, this is an undirected graph.

All the data has been anonymized to protect privacy. More details and background about the data are available here: http://snap.stanford.edu/data/egonets-Facebook.html (This website also has many other interesting datasets.)

1. **Loading the data (10 points)**
   The first step is to read the `facebook_combined.txt` file and load all the data into your Java classes. You can use either the Adjacency Matrix or Adjacency List as your data structure for storing the data.

2. **Implement Breadth-First Search (25 points)**
   Implement the BFS algorithm and run it on the data that you loaded in the previous step.

3. **Implement Depth-First Search (25 points)**
   Implement the DFS algorithm and run it on the data that you loaded in the previous step.

4. **Questions about the data**
   Answer the following questions using the data and the code that you implemented.

   a. What is the distance between nodes 10 and 1050? (5 points)
   b. Is the graph connected, i.e., is there a path from any given node *u* to any other given node *v*? Why? (5 points)
   c. Run the BFS algorithm starting from an arbitrary node. How many frontiers/steps does it need such that all nodes are visited? Repeat this three times starting from a random node every time. Does your answer change? Why? (5 points)
   d. How many nodes are within a distance of 4 from node 1985? (5 points)

Note 1: State any assumptions you make in the readme.txt file

Note 2: Overall, you can create as many (or as few) classes in Java that you like. Make sure to have a class with a `main` method that lets the user run the program and try out the different functionality.

Part 2 – Extra Credit (10 points)

For the normal credit part above, you used either an Adjacency Matrix or an Adjacency List for storing data. For Extra Credit, use the other one and compare how long the following take (in terms of time – seconds, milliseconds, etc.):

1. Loading data into memory
2. Running BFS
3. Running DFS

Explain, from a theoretical point of view, why the numbers are what they are.

For the EC part, you cannot have any help from the TAs/instructor.

Grading Criteria (for the programming part)

10% for compilation – If your code compiles, you get full credit. If not, you get a 0.
80% for functionality – Does the code work as required? Does it crash while running? Are there bugs? …
10% for style – Do you have good comments in the code? Are your variables named appropriately? ...

Programming – General Comments

Here are some guidelines wrt programming style for full credit.

Please use Javadoc-style comments.

For things like naming conventions, please see
http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

You can also install the Checkstyle plugin (https://checkstyle.org/eclipse-cs/) in Eclipse, which will automatically warn you about style violations.

Submission Instructions

We recommend submitting the theory part electronically also. However, you can turn in a physical copy at the start of class, if you prefer. The code should be submitted electronically. Please **do not** print it out.

In addition to the Java files (and the theory writeup), you should also submit a text file titled readme.txt, which should contain a short write-up about your software. How to run your program, any problems you experienced, etc. Think of the readme as a combination of instructions for the user and a chance for you to get partial credit.

Please create a folder called YOUR_PENNKEY. Places all your files inside this – the java files, theory writeup, the readme.txt file. Make sure to include answers to part 4 of the programming above. Zip up this folder. It will thus be called YOUR_PENNKEY.zip. So, e.g., my homework submission would be swapneel.zip. Please submit this zip file via canvas.