Part 2 - Programming

The project that was done was mostly implementation, but also included empirical analysis.

Description and Procedure

The first step was to collect our own data, since available Twitter data online were all anonymous. It included either just the topology of a part of Twitter, or only contained tweets tweeted by unreleased users. Therefore, we needed to retrieve data that suited our own needs.

We wrote a Java program that uses the Twitter4j library to access information from Twitter. We ran breadth first search beginning with a randomly chosen Twitter user. Since the scope of collecting data for all of Twitter was too large, BFS was most appropriate to get the topology data of a more local area. For each user, we collected a list of IDs of the user's followers and who the user is following. Then for all nodes that were not already in the queue or had already been visited, those users would be added to the queue.

Twitter's API had a rate limit for making requests to collect data, and as a result we were only able to make two requests per minute, one request being to get the IDs of the followers, and the other for the IDs of the following. We wrote the program so that it each execution of it would get the next node from the queue, and append the new data to the json file. We wrote a bash script that ran an executable jar file every minute, and we ran this script as a cronjob on the Eniac server. But even so, collecting a good amount of data within the time period was not easy, so we used 2 cronjobs to do so, with the other cronjob starting from a different node, namely Jack Dorsey.

Analysis

We could then do some analysis with the data, and also generalize these analysis methods to any set of Twitter data, so it can be used by any other user.

To run the analysis, execute `TwitterAnalysis.java` with at least one argument, with each argument being the name of the json file. To use our data set, enter the following in the command line after compiling the project:

```
java TwitterAnalysis adjacencyList.json adjacencyList_lh.json
```

After setting up the graph, the program will provide a few prompts for the user to perform analysis on the graph.

The following is a brief description of each class in the program:

- `analysis\TriadicClosure`
  Main implementation of the analysis for the graph. Has methods to find number of triangles in a graph, and find and rank appropriate friends to recommend for a particular user.
- `graph\Graph`
  Parses data from JSON files and populates a graph with Users.

- `graph\User`
  Represents a particular user on Twitter.
- `retrieval\GetUserNumber`
  Can be run while the data collection is being run to check how many users' data has been collected, and how many users are in the queue.
- `retrieval\RetrieveFriendsJSON`
  Retrieves user data, followers and following, for a particular user. The user is the next user ID on the queue. If it is the first time running, then it will initiate from a particular user.
- `retrieval\RetrieveTweets`
  Collects tweets for a particular user. Gets the next user ID from the list of nodes, and adds the tweet and hashtag data to the JSON file.
  (This class is not used due to the scope of collecting data for both the topology and the tweets, but can be used for further analysis and implementation.)

For curiosity's sake, the following is a link that can turn a user ID into the user handle: `http://tweeterid.com/`

Running the project
For this project's purposes, we have compiled an executable jar that can be run for the analysis of our Twitter data. Run it according to the following commands:

1.