CS 245 - Assignment 3 Spring, 2023

Assignment 3 - Friends in a Scandal

The goal of this assignment is to demonstrate your mastery of graphs, graph algorithms and object oriented design by handling a large dataset efficiently with respect to time and space.

Background

Enron was an American energy company based in Houston, Texas. It was founded in 1985 and was dissolved in 2001 due to an infamous scandal. As a result of a fraud investigation into the company's operations, a number of email messages sent and received by Enron employees has been made public. In this assignment, you will write an implementation for analysis on the data on Enron emails.

Functional Requirements

There are a number of requirements of this assignment, appearing in this section. Your implementation will be provided with a location to the uncompressed dataset as the first argument and an optional output file as the second argument. In other words, if your `main(...)` is located in your A3.java file, your implementation may be executed as follows:

```
java A3 /home/dbrizan/datasets/enron/maildir /tmp/connectors.txt
```

For this assignment, you are allowed to classes in the Java core (eg. String, HashMap) but no external or non-core classes. As always, your implementation must represent your own work. You must cite any portion of your source code which comes from an external source.

Requirement 1: Read the data file

Your implementation must read the valid mail files in the Enron dataset. The path to the uncompressed dataset will be the only argument to your implementation, as shown in the sample above. (The location of the same files on your machine will be different.) You can get a copy of the Enron dataset via https://www.cs.cmu.edu/~./enron/enron_mail_20150507.tar.gz. This dataset is large (1.7GB), and is provided as a compressed tar.gz file. On a Unix or MacOS system, you can uncompress it using the following command from a Unix shell or MacOS Terminal:

```
tar -xvzf enron_mail_20150507.tar.gz
```

If you need a smaller (incomplete) version of this data, contact the instructor.

The tar command will not remove the original file from your computer. However, the command will create a folder (`maildir`) and a number of subfolders representing an individual such as `symes-k` (Kate Symes

/ kate.symes@enron.com). Within each of these subfolders is the structure of the individual's email account and the email messages (files). By my count, there are 517402 valid mail files in this dataset and a total of 13053 individuals (where each unique email address represents an individual).

Do not add the Enron dataset to your repository or submit the dataset file with your implementation.

Your implementation will read all the valid mail files provided, store the necessary parts of those files and prepare for the other requirements in this section. You may notice that not all files in the folders are valid mail files.

You may represent the messages in the dataset as a *friendship graph*. A friendship graph is often an unweighted, undirected graph. The vertices in a friendship graph represent people, so may contain the name or email address of a person, among other details. Friendship graphs do not have self loops (an edge from a vertex to the same vertex), nor do they have multiple edges (more than edge between a pair of vertices). Note that the graph for this dataset may not be connected: there may be "islands" of people who are not connected to each other by an edge.

Requirement 2: Identify and print connectors

After reading the dataset, your implementation must identify *connectors*. In an undirected graph, vertex `v` is a connector if there are at least two other vertices `x` and `w` for which every path between `x` and `w` goes through `v`. For example, in the graph:

```
Liang ── Md ── Roberta
```

… if Md were removed from the graph, Liang and Roberta would no longer be connected. Md is therefore a connector.

It is possible to find all connectors in an undirected graph using DFS (the depth-first search algorithm), by keeping track of two additional quantities for every vertex `v`. These are:

- `dfsnum(v)` : This is the dfs number, assigned when a vertex is visited, dealt out in increasing order.
- `back(v)` : This is a number that is initially assigned when a vertex is visited, and is equal to dfsnum, but can be changed later as follows:
    - When the DFS backs up from a neighbor, w, to v, if dfsnum(v) > back(w), then back(v) is set to min(back(v), back(w))
    - If a neighbor, w, is already visited then back(v) is set to min(back(v),dfsnum(w))

When the DFS backs up from a neighbor, w, to v, if dfsnum(v) ≤ back(w), then v is identified as a connector, IF v is NOT the starting point for the DFS.If v is a starting point for DFS, it can be a connector, but another check must be made. You should construct an example to figure out details of this additional check.

Once your implementation identifies connectors, they should be printed to stdout (i.e. to the screen) and also to a file, the name of which is provided as the second argument to your implementation (`/tmp/connectors.txt` in the example above). If no file name is provided, your implementation should not output to a file and should only produce output to stdout.

Requirement 3: Provide details of each person

After reading the dataset and identifying connectors, your implementation must respond to user questions about individual email addresses. Specifically, when provided an email address in the dataset, your implementation must report:

- The number of unique email addresses to whom the individual sent messages
- The number of unique email addresses from whom the individual received messages
- The number of email addresses in the same "team" as the individual

An example of this interaction is as follows, with the implementation's outputs in bold and the user's three (3) inputs in non-bold font:

```
Email address of the individual (or EXIT to quit): kate.symes@enron.com
* kate.symes@enron.com has sent messages to X others
* kate.symes@enron.com has received messages from X others
* kate.symes@enron.com is in a team with X individuals
Email address of the individual (or EXIT to quit): notme@usfca.edu
Email address (notme@usfca.edu) not found in the dataset.
Email address of the individual (or EXIT to quit): EXIT
```

The contents of the files (above) may be used to determine the number of messages sent and received by each person. A "team" is any group of people who have exchanged email messages. In the example above, three people — Liang, Md and Roberta — have exchanged messages with each other. Despite the fact that Liang and Roberta have never exchanged messages with each other, they are on the same team because they may communicate through Md.

Grading

Your grade for this assignment will be determined as follows:

- 45% = Implementation: your class implementations must run successfully with the source files and data provided. It must produce the expected results. Any deviation from the expected results results in 0 credit for implementation.
- 25% = Efficiency: your code must consistently use the most efficient data structures for the task and must consistently use the most efficient algorithms on those data structures.
- 20% = Decomposition: in the eyes of the grader, your solution follows the suggestions above or otherwise must represent a reasonable object-oriented and procedural decomposition to this problem.
- 10% = Style: your code must be readable to the point of being self-documenting; in other words, it must have consistent comments describing the purpose of each class and the purpose of each function within a class. Names for variables and functions must be descriptive, and any code which is not straightforward or is in any way difficult to understand must be described with comments.

Assignments will not be accepted beyond the due date.

Submission

Check your Java code — including any main function, interfaces and classes — into a GitHub repository. Add any comments in the README.md file in the same repository as your source code. Submit the link to this GitHub repository on Canvas.