Assignment 1: Basics of Networks

Instructions for Submission

- Due date: Monday, September 10 at midnight on Canvas.
- You are free to use any programming language you are comfortable with. Some libraries that handle large scale graphs efficiently and provides a large number of functions for analyzing graphs are SNAP, igraph and networkX. We recommend using these libraries, but if you prefer others, please let us know such that we have them installed on the department computers for testing and grading.
- You can work alone or in teams of 2. By the end of the semester, you should have at least 3 (out of 5) assignments done alone.
- Submit only once per team. We will know your teammate from the submission (see below).
- What to submit: a tar file that includes the following:
  - A short pdf with:
    - your name(s) (this is the only way we will credit team members with the work done)
    - the results of your computations (nicely tabled), and
    - the plots required (if any).

    We do not want to read the text of the assignment in your pdf submission, but we need to understand what the numbers/plots you show mean. That is, use captions for pictures and tables, etc.
  - Your code and instructions on how to run your code in a text file. No need to include your names in the code, as we have this via your Canvas submission (see syllabus regarding FERPA rules, if you want). The name of the code file for problem 1 should be: `gen-structure.py` (or cpp or some other programming language). Similarly, the name of the running instruction file should be: `gen-structure.howtorun.txt`. The name of the code file for problem 2 should be: `graphic-check.py` (or cpp or some other programming language). Similarly, the name of the running instruction file should be: `graphic-check.howtorun.txt`. If there are any additional resources (e.g., libraries) that are required to run your code, please mention those in the how-to-run file.
- If any part of your code takes a long time to run (e.g., more than 10 minutes), report that in the how-to-run file with an estimate of running time and the specifics of the machine where you ran it (RAM, CPU and OS characteristics should suffice).
- References: Chapters 6 from Newman's "Networks—An Introduction", Chapter 2 from van Steen's "Graph Theory—An Introduction", and `snap.stanford.edu` for tutorials on SNAP.

Datasets

For Problem 1 use the following datasets and consider them all undirected. (Note that there are some explanations at the top of some of the files – remove them before you start testing. I left them there to help you check your results to the first tasks). You do not need the explanations below, but results might be more meaningful if you understand the networks you're analyzing:

1. `p2p-Gnutella04.txt`: Gnutella p2p network described here: `https://snap.stanford.edu/data/p2p-Gnutella04.html`
2. `as-caida20040105.txt`: AS from CAIDA as described here: `http://snap.stanford.edu/data/as-caida.html`
3. `wiki-Vote.txt`: `http://snap.stanford.edu/data/wiki-Vote.html`
4. `random5000by6.txt`: Generate your own graph as follows: generate a complete graph of 5000 nodes and consider only the nodes with node ids divisible by both 2 and 3. (The other nodes are thus removed, along with all their incidental edges.)

For Problem 2 use the datasets in Files/Datasets/HW1/DegreeSeq on Canvas.

For Problem 3 use the dataset in Files/Datasets/HW1/Visualization. Les Miserable is an undirected network which contains co-occurances of characters in Victor Hugo's novel 'Les Misrables'. A node represents a character and an edge between two nodes shows that these two characters appeared in the same chapter of the the book. The weight of each link indicates how often such a co-appearance occurred.

Problem Set

In these exercises, you will programmatically compute structural properties of 4 real-world networks, check the validity of the degree sequences of other 3 real-world networks, and experiment with graph visualization tool(s) to visualize a small network. All datasets are available on Canvas under Files/Datasets/HW1.

Problem 1: For each of the 4 datasets described above, write a program `gen-structure.py` (or cpp or some other programming language) to compute the following structural metrics. Your code should take the name of the file as its argument.

1. Size of the network:
   (a) Number of nodes: Your code should print a line:
   `Number of nodes in <graph name>: value`
   (b) Number of edges: Your code should print a line:
   `Number of edges in <graph name>: value`
2. Degree of nodes in the network:
   (a) Number of nodes which have degree = 1. Your code should print a line:
   `Number of nodes with degree=1 in <graph name>: value`
   (b) Node id(s) for the node(s) with the highest degree. Note that there might be multiple nodes with highest degree. Your code should print a line:
   `Node id(s) with highest degree in <graph name>: comma separated id(s) of nodes`
   (c) For the nodes with degree 1, compute the average degree of the nodes in the 1-hop neighborhood of its neighbor. Note that this means the average degree of the node's 2-hop neighborhood. For every node with degree 1, your code should print a line:
   `The average degree of <node-ID-with-degree-1>'s 2-hop neighborhood is: <value>`

(d) Plot of the degree distribution. Your code should create the plotted image in the same directory as your code and print:

`Degree distribution of <graph name> is in: <filename>`

3. Paths in the network:

(a) Approximate full diameter (maximum shortest path length) computed starting from 10, 100, 1000 random test nodes. Also calculate the average and variance across these 3 estimates of the diameter. Your code should print:

`Approx. diameter in <graph name> with sampling <number nodes> nodes: <diameter>`

and in the next line

`Approx. diameter in <graph name> (mean and variance): mean, variance`.

(b) Approximate effective diameter computed starting from 10, 100, 1000 random test nodes. Also calculate the average and variance across these 3 estimates of the diameter. Your code should print the following lines:

`Approx. effective diameter in <graph name> with sampling <number nodes> nodes: <diameter>`

and in the next line

`Approx. effective diameter in <graph name> (mean and variance): mean, variance`

(c) Plot of the distribution of the shortest path lengths in the network. Your code should create the plotted image in the same directory as your code and print:

`Shortest path distribution of <graph name> is in: <filename>`

4. Components of the network and of its complement:

(a) Fraction of nodes in the largest connected component. Your code should print a line:

`Fraction of nodes in largest connected component in <graph name>: value`

(b) Fraction of nodes in the largest connected component of the complement of the real graph. Your code should print a line:

`Fraction of nodes in largest connected component in <graph name>'s complement: value`

(c) Plot of the distribution of sizes of connected components for both the real graph and its complement. Your code should create the plotted image in the same directory as your code and print on stdout

`Component size distribution of <graph name> is in: <filename>`

`Component size distribution of the complement of <graph name> is in: <filename>`

Problem 2: You left a program running on your desktop at home, crawling the web to collect some network datasets for your research. Since your data collection is slowed down by the target site (e.g., Twitter API limits number of items collected per hour), you are also running some data preprocessing as you collect data. About 2 weeks in your data collection, when your program is about done collecting and storing the data locally, your clumsy roommate tripped (yet again!!!) on your power cord and your computer turns off. You are not sure the datasets were fully collected, and worse, you're worried about what happened to the data you stored in the file system. When you boot the machine back up (after you scolded your roommate at length and calmed down, resolving to buy some tape to glue your cord to the carpet and avoid disasters next time), you discovered that the only data files that are not corrupted (and thus you can read) are in a directory `DegreeSeq` and each file contains the vertex degree sequence of one of the networks you were collecting. While this

is really bad data loss, these files might still enable you to finish your research for the conference submission next week. But does the data really represent the degree sequence of a graph? (for example, maybe some of the node degrees were not written in the file before the crash, etc.) Check this for every file programmatically: that is, write a (set of?) program(s) or tools[1] in a file called `graphic-check.py` (or other extension) to check. The only input of your program should be a file with a graph degree sequence (already sorted). The output of your program should be:

`<filename> is a graphic degree sequence.` or

`<filename> is NOT a graphic degree sequence. It fails test X.`

where X can be your own text. Input datasets in Files/DATASETS/HW1/DegreeSeq.

Problem 3: The purpose of this exercise is to experiment with graph visualization tools. This experience might prove useful in future assignments or in your own research projects. The objective for this assignment is simple: visualize the best you can a smallish ('Les Miserables') network, include the best (most telling) picture you generated and describe in one paragraph what settings you tried and finally chose.

For this, you need to:

- Find and experiment with a tool (free and) useful for visualizing graphs. Examples: Gephi, Pajek, Ora, etc (many others exist).
- Run the tool on Les Miserables dataset. Generate the best pictures you can by tuning the tool parameters.
- Notice that many such tools have various graph metrics measurements available from their GUI. Make good use of them in the future.

[1] As always in this class, use tools that make your work most efficient. Some basic Unix commands might work well for some simpler input processing tasks, if you know how to use them. E.g., awk.