Social Computing [CS60017] 2024A

Assignment 1: *Measuring Network Properties*

Deadline for submission: 12th September 2024, EOD.

General Instructions

1. **This assignment is to be done individually by each student. You should not copy any code from one another, or from any web source. We will use standard plagiarism detection tools on the submissions. Plagiarised codes will be awarded zero for the entire assignment.**

2. This assignment is meant to use SNAP (http://snap.stanford.edu/) which is a library for handling very large graphs efficiently, and provides a large number of functions for analyzing graphs. It has two versions, one for C++ and another for Python. You are free to use either C++ or Python language. API Reference manual for both languages in available on their website.

3. Your codes should print out *exactly* what is asked, and in the specified format (sample given at the end). Do **NOT** output anything extra that isn't asked for. The results will be evaluated by an automated checker. There will be a **penalty of up to 20 marks** if the specified output format is not followed.

4. **How and what to submit:** Solutions should be uploaded via the CSE Moodle website (see course website for details). Submit one .zip or .tar.gz file containing a compressed folder that should contain all source codes, all files to be submitted (as per the task descriptions given below) and an instructions file (see next point). Name the compressed file the same as your roll number. Example: name the compressed file "21CS60R00.zip" or "21CS60R00.tar.gz" if your roll number is 21CS60R00.

5. Along with the source codes and files asked in the tasks, also submit an additional text file called "**instructions.txt**" where you should state how to run your codes as well as any additional information you want to convey, such as the version of Python or C++ compiler. If you have performed any optimizations for Problem 3, list them down. The instructions.txt file should also contain your name and roll number.

6. We should be able to run your submitted code in a computer with a reasonable configuration by following your submitted instructions. If any part of your code takes a long time to run (e.g., more than 5 minutes) report that in the instruction file with an estimate of time required.

7. Try writing codes into different functions and add several comments. Make sure your code can be easily read. **Illegible codes will be penalised**!

8. At the beginning of each of the source code files, set the seed of the random number generator of SNAP to **42**. For example, in Python:

```
import snap
Rnd = snap.TRnd(42)
Rnd.Randomize()
```

9. For any queries regarding the assignment, contact TA **Soham Poddar** (sohampoddar26@gmail.com).

‘

Problem 1: Dataset Preparation [10 points]

Download the edge list of graphs (the “combined.txt.gz” files) whose download links are given in Table 1. For each of the graphs we want to *generate the sub graphs* using the rules given in the same Table, to make them smaller for us to work with. These subgraphs will be used throughout this assignment. Write a program **gen_graphs.py** (or **.cpp**) to generate the edge list files (with “.elist” extensions) of the subgraphs. Also create a random network and a small-world network as described in Table 1.

Save all these graphs in a folder called **‘networks’**. Include these, but do **NOT** upload the original dataset files in your submission.

NOTE: *For this assignment, treat all graphs as undirected.*

Subgraph name	Rule to extract the subgraph	Link to download page
facebook.elist	Remove all nodes that have IDs divisible by 4	https://snap.stanford.edu/data/ego-Facebook.html
epinions.elist	Keep only the nodes whose IDs are divisible by 5, along with their signed weights	https://snap.stanford.edu/data/soc-sign-epinions.html

Subgraph name	Parameters for creating graph
random.elist	#Nodes: 1000, #Edges: 50000
smallworld.elist	#Nodes: 1000, Node Degree: 50, Rewire Probability = 0.6

Table 1: Rules for creating subgraphs for each of the networks and the submission filenames

Problem 2: Familiarizing with the SNAP Libarary [40 points]

Write a program **gen_structure.py** (or **.cpp**) to generate following structural metrics given below. Your code should take the path of the subgraph (specified in Table 1) as a command line argument. All the output needs to be printed on standard output, and not to any file. All plots generated should be kept in a folder called **"plots"**. We will run your code from a terminal with a the following command in Python and cpp (and similarly for the other subgraphs):
Python: `python gen_structure.py subgraphs/facebook.elist`
CPP: `./gen_structure subgraphs/facebook.elist`
NOTE: *For all fractions, round up to 4 decimal places while printing.*

1. **Size of the network** [4 points]

   (a) Number of nodes
   Your code should print the following line to stdout:
   Number of nodes: <value>

   (b) Number of edges
   Your code should print the following line to stdout:
   Number of edges: <value>

2. **Degree of nodes in the network** [6 points]

   (a) Number of nodes which have degree = 7
   Your code should print the following line to stdout:
   Number of nodes with degree=7: <value>

   (b) Node id(s) for the node with the highest degree. Note that there might be multiple nodes with highest degree
   Your code should print the following line to stdout:
   Node id(s) with highest degree: <comma separated id(s) of nodes>

   (c) Plot of the Degree distribution
   Your code should create the plotted image in the **"plots"** directory in a file named: **"deg_dist_<subgraph name>.png"**

3. **Paths in the network** [6 points]

   (a) Approximate full diameter (maximum shortest path length) starting from 1000 random test nodes.
   Your code should print lines in stdout:
   Approximate full diameter: <diameter value>

   (b) Approximate effective diameter computed starting from 1000 random test nodes.
   Your code should print lines in stdout:
   Approximate effective diameter: <diameter>

(c) Plot of the distribution of the shortest path lengths in the network.
Your code should create the plotted image in the **"plots"** directory in a file named: **"shortest_path_<subgraph name>.png"**

4. **Components of the network** [8 points]

(a) Fraction of nodes in the largest connected component
Your code should print the following line to stdout:
Fraction of nodes in largest connected component: <value>

(b) Number of edge bridges: An edge is a bridge if, when removed, increases the number of connected components.
Your code should print the following line to stdout:
Number of edge bridges: <value>

(c) Number of articulation points: A node is a articulation point if, when removed, increases the number of connected components.
Your code should print the following line to stdout:
Number of articulation points: <value>

(d) Plot of the distribution of sizes of connected components
Your code should create the plotted image in the **"plots"** directory in a file named: **"connected_comp_<subgraph name>.png"**

5. **Connectivity and clustering in the network** [10 points]

(a) Average clustering coefficient of the network (briefly explained here https://en.wikipedia.org/wiki/Clustering_coefficient#Network_average_clustering_coefficient).
Your code should print the following line to stdout:
Average clustering coefficient: <value>

(b) Number of triads
Your code should print the following line to stdout:
Number of triads: <value>

(c) Clustering coefficient of a randomly selected node. Also report the selected node id.
Your code should print the following line to stdout:
Clustering coefficient of random node <node id>: <value>

(d) Number of triads a randomly selected node participates in. Also report the selected node id.
Your code should print the following line to stdout:
Number of triads random node <node id> participates: <value>

(e) Plot of the distribution of clustering coefficient
Your code should create the plotted image in the **"plots"** directory in a file named: **"clustering_coeff_<subgraph name>.png"**

6. **Centrality metrics of the network** [6 points]

   (a) Degree centrality
   Your code should print the following line to stdout:
   Top 5 nodes by degree centrality: <space separated list of top 5 node ids>

   (b) Closeness centrality
   Your code should print the following line to stdout:
   Top 5 nodes by closeness centrality: <space separated list of top 5 node ids>

   (c) Betweenness centrality
   Your code should print the following line to stdout:
   Top 5 nodes by betweenness centrality: <space separated list of top 5 node ids>

Sample Output

The output of your code to stdout should be exactly in the format shown below (the numbers below are random).

```
Number of nodes: 42
Number of edges: 42
Number of nodes with degree=7: 42
Node id(s) with highest degree: 1,3,4
Approximate full diameter: 52
Approximate effective diameter: 50
Fraction of nodes in largest connected component: 0.7
Number of edge bridges: 42
Number of articulation points: 42
Average clustering coefficient: 0.5
Number of triads: 42
Clustering coefficient of random node 3: 0.5
Number of triads random node 3 participates: 42
Top 5 nodes by degree centrality: 15 14 13 12 11
Top 5 nodes by closeness centrality: 25 24 23 22 21
Top 5 nodes by betweenness centrality: 35 34 33 32 31
```

Problem 3: Compute Centrality Metrics [50 points]

Write a program **gen_centrality.py** (or in cpp) to compute the following centrality metrics for a graph. Your code should take the path of the subgraph (specified in Table 1) as a command line argument. It should output a text file each for the centrality measures, which contains a line for each of the nodes. Name the output files "closeness.txt", "betweenness.txt" and "pagerank.txt". Generate the files inside a folder called "**centralities**". Each line in the output files has the format: *nodeID <white space> centrality value*.
The centrality values can be rounded to **4** decimal places. The nodes in each file should be sorted by the centrality value. (Sample at the end) In the **instructions.txt**, write 2-3 brief points per centrality, discussing the design decisions you had to take with justifications, highlighting any novel methods used.

1. **Closeness centrality** for node i, given by $C_i = \frac{n-1}{\sum\limits_j d_{ij}}$, where d_{ij} is the length of the shortest path from i to j, and n is the number of nodes in the graph. [10 points]
2. **Betweenness centrality** for node i, given by $B_i = \frac{2}{(n-1)(n-2)} \sum\limits_{st} \frac{n_{st}^i}{g_{st}}$, where n_{st}^i is the number of shortest paths between nodes s and t which pass through i, and g_{st} is the total number of shortest paths between nodes s and t. [18 points]
3. **Biased PageRank** for a node i calculated using the standard PageRank power-iteration method (as discussed in class). Use a non-uniform preference vector biased towards nodes with higher average incoming scores. The incoming scores of a node are the signed weights of its incoming edges. For *Epinions* dataset, these should be the given signed weights and for other datasets, these should all be 1. Use damping factor, $\alpha = 0.8$. [20 points]

Finally, use the 3 output files generated and calculate how many 'influencer' nodes are present in a given network. There can be various ways of identifying influencers, but for this assignment, consider those nodes that are among the top 200 nodes using all three centrality metrics. [2 points]

Your code should print a line in stdout:
"Number of influencer nodes: <value>"

Please note the following carefully for Problem 3:

1. You can build your code on SNAP (using graph classes etc.). There are already built-in functions in SNAP to calculate all these centralities (as you used for the previous problem). However, you should **NOT** use these functions for this problem. You need to implement these centralities yourself, as extensions to SNAP's graph framework. The signed weights in the epinions graph is required only for the PageRank part, and need not be loaded when performing other subtasks. You can write your own custom code to handle/load the signed weights into a directed graph.

2. For calculating closeness centrality for nodes in disconnected graphs, you may apply different strategies. Here are some examples here but you can try some other strategies too. **(a)** Taking the distance for "unreachable" nodes as N (number of nodes) [used in the snap library]; **(b)** Instead of $(N-1)/sum(dist_{ij})$ you can use $sum(1/dist_{ij})$; **(c)** Normalize with $(N_{comp}-1)$ instead of $(N-1)$, where N_{comp} is the number of nodes in the same connected component as the given node.

3. In fact you can implement more efficient ways of computing centralities in your code. There are multiple algorithms, like variation of Floyd Warshall algorithm, Johnson's algorithm and Brandes' algorithm. You can read more about them here: https://en.wikipedia.org/wiki/Betweenness_centrality#Algorithms.

4. For computing betweenness and the preference vector for PageRank from the weights, you can use your heuristics / strategies to optimize your code for speedup as well as obtaining "desirable" rankings. Theres no one right way to do these. (what constitutes as desirable rankings is for you to think and explore)

5. Try writing codes into different functions and add several comments. Make sure your code can be easily read. Illegible codes will be **penalised**!

Sample Centrality file

The centrality files should have the format shown below (the numbers below are random). There should be a line for each of the nodes in the network, sorted by the centrality value.

```
5    0.6667
1    0.5
3    0.152
4    0.3333
```