CX 4010 / CSE 6010
Assignment 5
Graph Analytics

Due Dates:

- Due: 11:00 AM, Friday, October 11, 2019
- Revision (optional): 11:59 PM, Monday October 14, 2019
- No late submissions will be accepted

1. Background

Graphs arise in many applications in science, engineering, and the social sciences. Here, we are particularly interested in a class of graphs known as *scale-free networks*. An important characteristic of scale-free network is they include some number of *hub* nodes that contain a relatively large number of links, but the vast majority of nodes contain relatively few links. The number of links attached to a node is referred to as the node's *degree*. Technically, a scale-free network is defined as a network whose node degree distribution follows a power-law. Scale free networks are interesting because there is empirical evidence that they arise in many applications of current interest. See [1] for information on power law distributions and applications. Here, we will assume all graphs are undirected graphs.

You must use the adjacency list approach to represent your graph in memory. The adjacency matrix representation does not scale well to large graphs so should not be used.

This assignment involves generating scale free network graphs and performing an analysis of the maximum path length through the graph. The problem is divided into two parts. The first part involves writing a program to generate a graph, storing it in a file, and considering if the generated topology is scale-free. The second involves writing another separate program that reads the graph file created by the graph generation program and computing the diameter of the graph to evaluate how the maximum distance between nodes increases with network size.

You will work in teams of two persons each to attack this problem. For each team one person will develop a program to generate a network graph and completing associated tasks. The second person will develop a program that reads the graph file and performs an analysis of the graph. The two programs should be completely separate "stand-alone" programs. The interface between the two programs will be the format of the file specifying the graph. You and your partner must define and document a common file format for the graph, and write up all of your results into a single, combined report. While most teams have two people, one may have three. If you are part of a three-person team, each person on the team will implement one of the two programs, so your team will have two separate implementations of one part and one implementation of the other, and the three of you should jointly complete the remaining tasks and write the report.

2. Part 1: Random Graph Generation

To complete this part of the assignment, write a program called `graphgen` that creates a scale-free network graph, stores the result into a file, and generates a histogram of the node degree distribution. Your program can have up to five command line parameters. The first three are required, and the other two are optional. The first parameter is the number of nodes in the network to be generated, and the second and third are floating point values used to determine link weights as described below. The two optional parameters may appear in any order in the command line, and indicate the name of the file where the resulting topology is to be written, and the name of another file where the histogram of the node degree distribution is to be written, if one is desired. For example, the command line (in Unix):

```
% graphgen 2000 0.5 1.5 -o top.txt –h histo.txt
```

executes the `graphgen` program (which you will write) to creates a scale-free network topology containing 2000 nodes and link parameters 0.5 and 1.5. The program writes the resulting topology into a text file called `top.txt`; the `-o` flag in the command line indicates the name of the output file follows. If no such file is specified as a command line argument, i.e., no `-o` flag appears, the output should be written into a file called `topology.txt`. Similarly, the `-h` flag indicates a histogram is to be created and written into a file whose name follows the `-h` flag, or in this example, `histo.txt`. If the `-h` flag is not present, no histogram is generated.

The network topology is constructed using a well-known principle called *preferential attachment* [2]. The key idea behind preferential attachment is when a node is added to an existing network, it is more likely to establish a link to another node that already has a high degree (a large number of edges) compared to nodes with fewer edges. This construction mechanism is also referred to as "the rich get richer" in the sense that the "richness" of a node is based on its node degree.

More precisely, when a new node is added to an existing network, it is attached via an edge to a randomly selected node *i* with probability *p(i)*. *p(i)* is defined as: *d(i) / D* where *d(i)* is the degree of node *i*, and *D* is the sum of node degrees of all nodes in the existing network prior to adding the new node.

The algorithm to generate the topology with *N* nodes is as follows:

1. Construct a fully connected topology containing n_0 nodes, where $n_0 << N$. A fully connected network means there is a single edge between every pair of nodes (so a fully connected network with n_0 nodes has $n_0*(n_0-1)/2$ edges). For this assignment, assume n_0 is equal to 3 nodes.
2. Assuming the network now contains *k* nodes (labeled *0, 1, 2, ... k-1*), add one additional node *k* and one edge between *k* and one randomly selected node *j* with probability *p(j)* as defined above using the preferential attachment rule.
3. Repeat the previous step until the network contains *N* nodes.

Your program must include at least three functions:

1. A function that creates the network and stores it in memory
2. A function that outputs a histogram of the node degree distribution for the created network to a file; each line of the file should contain three values: `i, num, list,` where `i` is the node degree (1, 2, 3, …), `num` is the number of nodes in your network with degree `i`, and `list` is a list of nodes with degree `i`. The first line of the generated file should have information for nodes of degree 1, the second line has information for nodes of degree 2, etc.
3. A function that writes the network topology stored in memory into a file in the format agreed to with your partner.

Your program for generating the graph must also determine a weight for each link in the graph, and include this information in the topology file that is created. The link weight should be a floating point (type `double`) random number uniformly distributed in the interval `[w1, w2]` where w1 and w2 are the command line parameters indicated earlier, and `w1` ≤ `w2`. Note the link weight parameters must be strictly positive and it is possible `w1` and `w2` are the same value.

You should complete an analysis of the topologies generated by your program and argue that the created network is indeed scale-free. In this plot, the horizontal axis is the node degree, and the vertical axis indicates the number of times a node with that degree appeared in the network you generated. An easy way to qualitatively determine if the node-degree histogram follows a power law is to create a log-log plot of the node degree histogram, i.e., plot `log(d)` on the horizontal axis, and `log(c)` on the vertical axis, where `d` is the degree, and `c` is a count of the number of nodes with degree `d`. If the distribution follows a power law, the plot should produce (approximately) a straight line. Do this for as large a network as your program can generate in a reasonable amount of time, say a few minutes.

In addition to documenting your software you will need to define the format of the file containing the graph. This file format defines the interface between the graph generation and the graph analysis program. The file format should be clearly documented in your report.

3. Part 2: Network Analysis

We are interested in understanding how the path length between vertices grows as the size of the scale-free network grows. The metric of interest here is called the network *diameter*. Diameter is defined as follows. A path from a source to a destination vertex is a sequence of edges leading from the source to the destination. The length of this path is the sum of the weights assigned to the edges making up the path. The *distance* between any two vertices in a graph is the length of the *shortest* path between those two vertices. The diameter of the graph is defined as the maximum distance among all pairs of vertices in the graph. The objective of this part of the assignment is to understand how the graph diameter grows as *N* is increased in a scale free network with *N* vertices. Here, you will assume that each edge of the network is bidirectional with weight specified in the topology file input to the program.

The graph analysis program reads a file containing the graph topology (produced by the graph generation program) and computes the diameter of the graph. To compute the diameter, use Dijkstra's algorithm to compute the minimum length path between a source vertex *s* and every other vertex in the network. Write your program to be as efficient as possible. You may reuse code developed earlier in the semester for the priority queue, but be sure to cite the source of this software in your report and in the software itself.

Your program should be called `analysis`. Use the following command line to execute your program:

```
% analysis topology –o output
```

The first argument specifies the name of the file that holds the topology of the network you are analyzing. The second (`-o output`) is optional. If specified, it indicates the name of the file your program will create to hold the output. The output file should provide the following information:

- The first line indicates the diameter of the network that was analyzed
- The next *N* lines contain information on each of the *N* nodes. Specifically, each line should contain (1) the node number `s`, and (2) the maximum distance from node `s` to any other

node in the network, and (3) a list of the nodes forming the path for this maximum distance from node `s`. If multiple paths have this maximum distance, you need only output one such path.

If the output file is not specified in the command line via the `–o` flag, the program should simply print the computer network diameter to the screen.

The report for this part of the assignment should plot the size of the network diameter as a function of *N*. Use as large a value of *N* as your program can process in a reasonable amount of time (e.g., a few minutes). Use a link weight of 1.0 for all links for this plot, i.e., `w1=w2=1.0`. Note, however, your program must correctly compute the diameter for any valid set of link weights.

4. Report

A report is required that describes the software, the results produced by the software, as discussed above, and your team's analyses. Although there are two distinct parts to this assignment, your report should be written as one document, not two separate documents. Be sure to include proper citations and a bibliography in your report.

As part of the report, perform a brief literature search on scale free networks, focusing on the applications where scale-free networks are believed to arise. Describe briefly one such application, and cite relevant literature that provides evidence that scale free networks arise in this application. The network need not exactly match the preferential attachment model, but there should be empirical evidence suggesting the topology has properties such as hub and leaf nodes. What do the nodes and links represent in this application, and what kinds of questions might a graph analysis answer? Argue why the preferential attachment rule might be appropriate for this application.

Similarly, search the literature for information on the expected diameter of scale free networks. Compare your results with what is expected, as described in the literature. Write up your findings in the report. Don't forget to include proper citations to the literature as appropriate.

5. Final Comments

Turn in your report, software, and other files (e.g., sample output) as a single zip file. Your software must be well documented and include comments so the code is easy to understand. You should include a README file with instructions on how to compile and run your program.

Each part of this assignment has multiple parts. For instance, you will need code to process command line arguments and generate outputs, as well as code to create and analyze the graph. We recommend you develop the software incrementally rather than develop the complete program all at once. Think through what functions you'll need at the start and define their prototypes. You might want to discuss the design with your partner so together you can identify flaws or other issues. Then you can implement the functions piece by piece to build up the complete program. For example, you might write the code to manipulate the data structures representing the graph first and get this code to work before implementing other parts of the assignment.

We encourage you to use the web but be careful not to utilize code copied directly from the web. You must adhere to the Georgia Tech honor code, and the collaboration policy stated in the course syllabus. Specifically, you are encouraged to discuss the problem and possible solutions with other students (as well as the TA/instructor), however, all code that you turn in must be completely your own work. Disseminating your code to other students outside your team is strictly prohibited.

6. References

[1] M.E.J. Newman, "Power Laws, Pareto Distributions and Zipf's Law," *Contemporary Physics*, Vol. 46, No. 5. (1 September 2005), pp. 323-351, doi:10.1080/00107510500052444

[2] Barabasi, A.L. and R. Albert, *Emergence of Scaling in Random Networks.* Science, 1999. **286**(5439): p. 509-512.