Emily Forman

SNAP - EU Email Connection Network - https://snap.stanford.edu/data/email-EuAll.html

For my DS210 final project, I used the data set (linked above), which included email data from a European research institution during an 18 month period in October 2003 to May 2005. In this context, the nodes refer to the email addresses of the users, and the edges exist if there is an email sent between two email addresses, and there are 265,214 nodes and 420,045 edges. Of course this is an extremely large data set, so I set a max limit number of nodes to 500 to reduce run time for the purpose of grading and displayed that output, but over the course of this project I ran it with a max limit of 1000 nodes. This project implements "6 degrees of separation," in which I utilized the Breadth-First Search and shortest path algorithms to essentially determine which nodes had the highest amount of incoming edges, and therefore were most significant to the network of emails within this research data.

My code is split into 3 modules; firstly, bfs.rs contains all the functions defined to perform Breadth-First Search, data.rs used to read the data from my downloaded data set linked above, and main.rs to find the average shortest path length and perform the tests.

bfs.rs module

More specifically, in the bfs.rs module, three functions are named. The first function is "shortest_path," which initializes the hashmap graph of the starting node, which I made the distance to be 0, to the nodes that are visited. A loop is then created, and continues while nodes are in the initialized queue, and then dequeues the current node from the front of the queue. If the current node is the end node, it will store the distance, and if not, the loop will go through each neighbor and add it to the queue so that it is considered to be visited and stored. The next function, "shortest_path_length" involves a nested for loop that uses start and end to define the

node pairings. When the shortest path is found, it's added to “totallen” (also increasing “numberpaths” by 1 at this point). The average length is then computed through this iteration over the pairs of nodes.. Following this, the next function named “top5nodes” iterates through the nodes and edges. For each node in the hashmap, the number of incoming edges is derived. I implemented “sort_unstable_by” so that the nodes are in descending order to be able to print the nodes with the highest incoming edges. After computing the count of edges, it will print the top 5 nodes with the highest amount of edges. The final function “percent_nodes” is another way I decided to analyze the significance of certain nodes as I was able to verify that the highest amount of incoming edges is a small percentage of the overall nodes. This, therefore, shows the distribution among the different nodes, to provide a better understanding and visualization of this data set.

data.rs module

In the data.rs module, I was able to download the data set from the website and use this to make the function to read it from my file. This is done by initializing an empty Hashmap, Hashset for the graph of nodes and their neighbors and populating the graph.

main.rs module

In the main.rs module, essentially the functions from bfs.rs and data.rs are being utilized to read the dataset, calculate the average shortest path length, print the top 5 nodes with the highest number of incoming edges, and print the percentage of nodes with particular ranges of incoming of nodes. I included three tests, first validating that the total number of nodes is 265,214 as given in the website, and the second validating the read function by showing that the graph is correct by initializing a temporary path, and comparing against the expected graph.

The third verifies that the total percentages of the distribution of nodes to range number of edges is 100%, to ensure that it is inclusive of all of the data used. My outputs from running the code (cargo run) and the tests (cargo test) are:

```
running 3 tests
test test_percent_nodes ... ok
test test_read ... ok
test test_node_count ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.89s

(base) emilyforman@crc-dot1x-nat-10-239-29-28 EFds210project %
```

```
The top 5 nodes with the highest number of incoming edges are:
1. Node 314: with 63 incoming edges
2. Node 10: with 53 incoming edges
3. Node 192: with 51 incoming edges
4. Node 240: with 43 incoming edges
5. Node 175: with 42 incoming edges
The percentage of nodes with a range of 0 to 10 edges: 77.88%
The percentage of nodes with a range of 11 to 20 edges: 14.45%
The percentage of nodes with a range of 21 to 30 edges: 4.51%
The percentage of nodes with a range of 31 to 40 edges: 1.58%
The percentage of nodes with a range of 41 to 50 edges: 0.90%
The percentage of nodes with a range of 51 to 60 edges: 0.45%
The percentage of nodes with a range of 61 to 70 edges: 0.23%
The percentage of nodes with a range of 71 to 80 edges: 0.00%
Average shortest path length: 3.27
(base) emilyforman@crc-dot1x-nat-10-239-29-28 EFds210project %
```

As shown in the output, the top 5 nodes and their number of edges are printed as well as the average shortest path length of 3.27. This indicates that this communication network is well connected and efficient. If this average shortage path length was a larger number, more steps would be required to reach from node to node, and the overall connectivity would be much less. Also, the output stated the percentage of nodes pertaining to specific ranges of number of edges, showing that the distribution overall. I can conclude that the majority of the nodes (email addresses) fall in the range of having 0 to 10 incoming edges, which displays a lower level of significance in comparison to the very small percentage of nodes that have the highest number of edges incoming. These top 5 nodes have the most incoming edges, or in this context, these are

the email addresses with the most incoming emails. They are likely much more influential through this whole network of EU email addresses and ultimately are the epicenter of the system.

I found this data set very interesting to work with because it is so applicable to the real word connection systems. I was able to follow along with the implementation of difficult algorithms with logic, as some email addresses will receive more emails than others. The most interesting part of this is correlating potential reasoning to why certain emails have many more incoming nodes than others, and that since the average path distance was a small number, there was a strong interconnectivity among this communication network!

Resource:

https://www.sotr.blog/articles/breadth-first-search , lecture notes, rust documentation from official rust website.