Patrick Soong

DS210

Collaborators: None

# Final Project Report

**Overview:**

For my DS210 final project, I analyzed three undirected graph datasets focusing on social networks from Stanford's Large Network Dataset Collection. I used two of my txt files from data from Deezer, where one focused on users in Croatia and the other in Hungary. Additionally, I also used a dataset that analyzed users from github. My project can serve as a user analysis tool that companies can use in order to research how interlinked their networks are. The datasets can be found here: https://snap.stanford.edu/data/gemsec-Deezer.html, https://snap.stanford.edu/data/github-social.html)

**Graph.rs Module**

To start, I created a function to read my file that takes a file path and a number as parameters. The graph is implemented as an undirected graph, meaning that the edges have no specific direction. If A is connected to B, then B is also connected to A. I then created an "add_edge" method which allows adding edges between two nodes in the graph, this ensures that the graph is undirected by adding both nodes as neighbors of eachother in the 'edges' hashmap. Finally, I created a 'neighbors' method, which takes in the ID of a node in the graph and returns a vector containing the user IDs of all neihgboring nodes connected to the specified node. This is useful later for implementation of algorithms.

**Algos.rs Module**

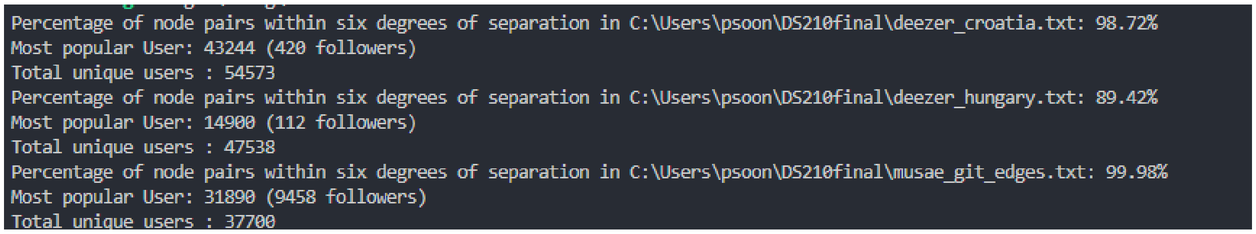
This module focuses on the algorithms that I'll be using for my project. First, I wanted to focus on the theory of 6 degrees of separation as I believed that it is important especially for a social network. To do this, I used the breadth-first search method, which starts from a given vertex and returns a hashmap containing the distances from the start to other vertices. This function continues a process of going through a "queue", where each neighbor that is not already in the 'degrees' hashmap getst inserted with a degree one greater than the degree of the dequeued node. Once this finishes, then the 'degrees' hashmap is returned.

My second function served to analyze the node with the most followers, which was implemented through finding the max degree and the node associated with it by iterating through the graph's edges using a for loop. It then calculates the degree for each node and its neighbor and returns a tuple containing the node ID with the most followers.

Lastly, I made a function to calculate the total number of unique users, which initializes a hashset to store unique user ids by iterating through the edges in the graph.

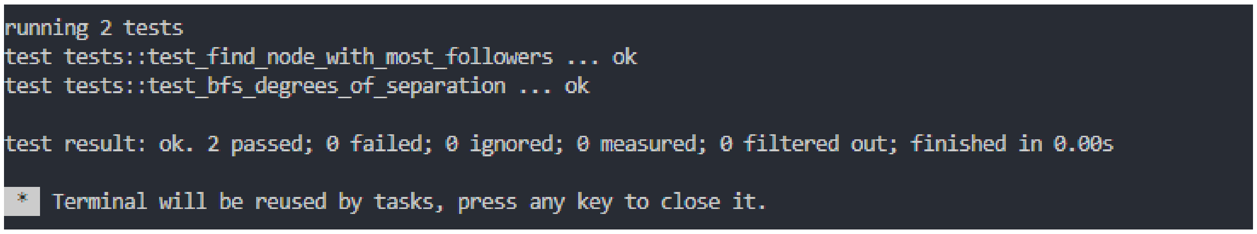
**Main**

I wanted to see how these different datasets compared to eachother, so I made a loop iterating through each txt file to calculate the percentage of users within 6 degrees of separation, the most popular user, and the total unique number of users. Here are my results:



```
Percentage of node pairs within six degrees of separation in C:\Users\psoon\DS210final\deezer_croatia.txt: 98.72%
Most popular User: 43244 (420 followers)
Total unique users : 54573
Percentage of node pairs within six degrees of separation in C:\Users\psoon\DS210final\deezer_hungary.txt: 89.42%
Most popular User: 14900 (112 followers)
Total unique users : 47538
Percentage of node pairs within six degrees of separation in C:\Users\psoon\DS210final\musae_git_edges.txt: 99.98%
Most popular User: 31890 (9458 followers)
Total unique users : 37700
```

My test cases focused on testing my bfs function and the function finding the most popular user, with results below:



```
running 2 tests
test tests::test_find_node_with_most_followers ... ok
test tests::test_bfs_degrees_of_separation ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

 *  Terminal will be reused by tasks, press any key to close it.
```

Overall, I found these results to be really interesting, as it proves the theory of the 6 degrees of separation in a real-world sense and is also applicable to different types of applications and even geographic locations.

**Resources:**

[https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm#:~:text=Breadth%2DFirst%20Search%20Algorithm%20or,the%20next%2Dlevel%20neighbor%20nodes](https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm#:~:text=Breadth%2DFirst%20Search%20Algorithm%20or,the%20next%2Dlevel%20neighbor%20nodes).