Youssef Souayah
CDS DS 210: Programming For Data Science
Final Project Report
May 5, 2024
Repository Link: https://github.com/ysouayah/DS210
Dataset: https://snap.stanford.edu/data/ego-Facebook.html

Introduction
My project aims to analyze social network data using graph theory and network analysis techniques. I specifically wanted to use different metrics to prove the small-world hypothesis - that every person on Earth is connected to everyone else. The dataset used in this project represents connections between individuals on Facebook, where nodes represent User IDs, and edges represent connections (friendships) between them. The analysis includes calculating average shortest path length, average distance between nodes, identifying similar and dissimilar users, and examining the degree distribution of the network. It's important to note that this is slightly different from my original plan because I wanted to use a Twitter dataset but it was too big for my computer to run without cutting down on the number of edges the program was analyzing.

fn read_data_file
I first made the read_data_file function to clean and read the data file. It expects the file to contain pairs of integers representing connections between users. Each line in the file represents an edge in the graph. To put it simply, the code mainly checks for whitespace and any missing values. After parsing through the data and collecting the relevant information I wish to analyze, it creates the edges using two nodes per edge. The function then returns a vector of tuples, where each tuple represents an edge.

fn jaccard_similarity
To begin, I first wanted to find if there are any outlier edge pairs. These pairs could either be having the exact same set of friends, potentially indicating it is the same individual under two users, or they have no friends in common. The jaccard_similarity function calculates the Jaccard similarity coefficient between two edges. It is a measure of how similar the sets are to each other. It computes the size of the intersection divided by the size of the union of the sets.
The find_similar_dissimilar_users function also works to identify the most similar and dissimilar pairs of users based on their Jaccard similarity coefficients. It constructs an adjacency list representation of the graph, computes Jaccard similarities for all pairs of users, and finds the pairs with the highest and lowest similarities. The code then prints out two lines, one for the pair with the similarity closest to 1, and the other with the similarity closest to 0.

fn calculate_average_shortest_path_length
My main focus for the project was to analyze the average shortest paths between nodes. This would indicate how close the nodes are to each other. If the average distance is less than 1, that would support the small-world hypothesis. The calculate_average_shortest_path_length function calculates the average shortest path length in the graph using Dijkstra's algorithm. It

computes the shortest path between all pairs of nodes in the graph and then averages these path lengths to find the average shortest path length.

fn find_similar_dissimilar_users
Another indicator of how closely connected people are on Facebook is to find the degree distribution, which is performed within the find_similar_dissimilar_users function. It calculates the degree distribution of the graph, which represents the distribution of the number of friends or connections among users. The function prints the count of users having a specific number of friends, categorized into different ranges. To keep the code compact, I divided the outputs across different degree ranges: under 10, between 10 and 25, and over 25. I also did this because if the higher degree correlates to having a higher number of nodes, that would support the small-world hypothesis. If one were curious about what the entire data looks like, I added a print statement before the for loop. Detailed instructions for what lines to omit are found in the repository.

Running the project and its output
To run the project, you need to specify the path to the social network data file in the main function. This will be different across devices. After reading the data, it first outputs the average shortest path. Throughout all my tests, the shortest path was always ⅓ suggesting that the Facebook users in the dataset are extremely interconnected. Since the average shortest path between users is consistent throughout, this supports the small-world hypothesis.

The code then outputs the two user pairs with the most and least Jaccard similarity. One flaw I realized with this is that there are a lot of user pairs with similarity at either 1 or 0 so the output is something different every time. Attempting to clean the data further and remove these pairs, however, causes issues down the line with the degree distribution so I decided to keep them for simplicity and because I hypothesize that removing those values could impact the lower-end of my degree distribution (mostly around the 1-10 range).

The next output was the degree distribution. Generally there is an increase in the number of people with higher degrees, which correlates to having more friends. If given more time I would look for a better cut off to keep the output compact, but as it stands, the fact that the highest number of users have over 25 friends also supports the small-world hypothesis.