My Rust project is designed to analyze a dataset from Facebook, which is "facebook_combined.txt" in my code. This dataset is derived from Facebook and represents a social network. It includes node features (profiles), circles, and ego networks. This is the breakdown of the components:

- **Nodes**: These represent individual Facebook users. There are **4,039** nodes in the dataset.
- **Edges**: These represent connections between users, i.e., friendships. The dataset contains **88,234** edges.
- **Node Features**: These are characteristics or attributes of the nodes. While the dataset includes these features, their interpretation has been obscured for privacy. For example, a feature like "political=Democratic Party" in the original dataset might be represented as "political=anonymized feature 1" in this dataset.
- **Circles/Friends Lists**: These are groups of nodes that are connected to each other more closely than to the rest of the network.
- **Ego Networks**: These are subnetworks centered around a single node (the 'ego'). The ego, its direct connections (the 'alters'), and all the connections among the alters make up the ego network.

In this project, I'm using facebook_combined.txt.gz, which contains all the edges from all ego networks combined. For this dataset, each line represents a connection ('edge') between two users ('nodes')on Facebook.

- The first number on each line represents a node in the network.
- The second number on each line represents another node that is connected to the first node.

The main goal of this code is to analyze a social network graph and to explore the concept of "Six Degrees of Separation" in a social network. First it begins by importing necessary modules and declaring two modules, data and graph. The load_facebook_data function reads a file containing HashMap where each key is a node (a String) and the value is a Vec<String> representing the neighbors of the node. The function reads the file line by line, splits each line into two parts (representing two nodes that are connected), and adds the connection to the graph.

The main function is the entry point of the project. It loads the social network data from a file named "facebook_combined.txt" and performs several analyses on it:

- It first calculates the average distance between the selected nodes using the six_degree_of separation function
- Then it calculates the average and standard deviation of the highest degree (the number of connections a node has) in the graph using the average_and_std_dev_of_highest_degree function.
- And lastly, it counts the number of nodes separated by one, two, three, etc. Degrees for each degree in the graph using the nodes_separated_by_degree function.

→I used chat gpt to find why I have errors and to fix them.

The graph.rs is used for social network analysis, where nodes represent people and edges represent relationships between them. The different functions are:

- **Breadth-First Search (BFS):** The bfs function performs a breadth-first search on the graph starting from start_node. It keeps track of all visited nodes to avoid revisiting them. If a visited node is in selected_nodes, it increments the distance variable. The function returns the total distance.
- **Six Degrees of Separation:** The six_degrees_of_separation function calculates the average distance between all pairs of nodes in selected_nodes using the bfs function. It sums up the distances and divides by the number of nodes in selected_nodes.
- **Average and Standard Deviation of Degree:** The average_and_std_dev_of_highest_degree function calculates the average and standard deviation of the degrees of the nodes in the graph. The degree of a node is the number of its neighbors.
- **Nodes Separated by Degree:** The nodes_separated_by_degree function creates a HashMap where each key is a degree and its value is a vector of counts. For each node in the graph, it increments the count at the index corresponding to the degree of separation.

→I used chat gpt to find why I have errors and to fix them.

In the data rs., the function named random_nodes selects a specified number of nodes from a graph at random, but with a probability proportional to their degree (the number of their neighbors). This is the breakdown of what each part of the function represents:

- **Function signature:** The function takes two arguments: a reference to a HashMap representing the graph and an integer representing the number of nodes to select. It returns a HashSet containing the references to the selected nodes.
- **Collect nodes and weights:** The function first creates a vector of all the nodes in the graph and a corresponding vector of their weights. The weight of a node is the number of its neighbors, which is used as the probability in the random selection process.
- **Random Number Generator:** It initialized a random number generator(rng) using the rand::thread_rng() function.
- **Weighted Distribution:** It creates a weighted distribution(dist) using the WeightedIndex:: new function with the weights vector. This distribution will return indices from the all_nodes vector with a probability proportional to the corresponding weight.
- **Select Nodes:** It then enters a loop where it continues to select nodes at random from the all_nodes vector using the weighted distribution until it has selected the specified number of nodes (num_nodes). The selected nodes are inserted into the selected_nodes HashSet.
- **Return Selected Nodes:** Finally, it returns the HashSet of selected nodes.

→ I used chat gpt to find why I have errors and to fix them.

The degreedistribution.rs:

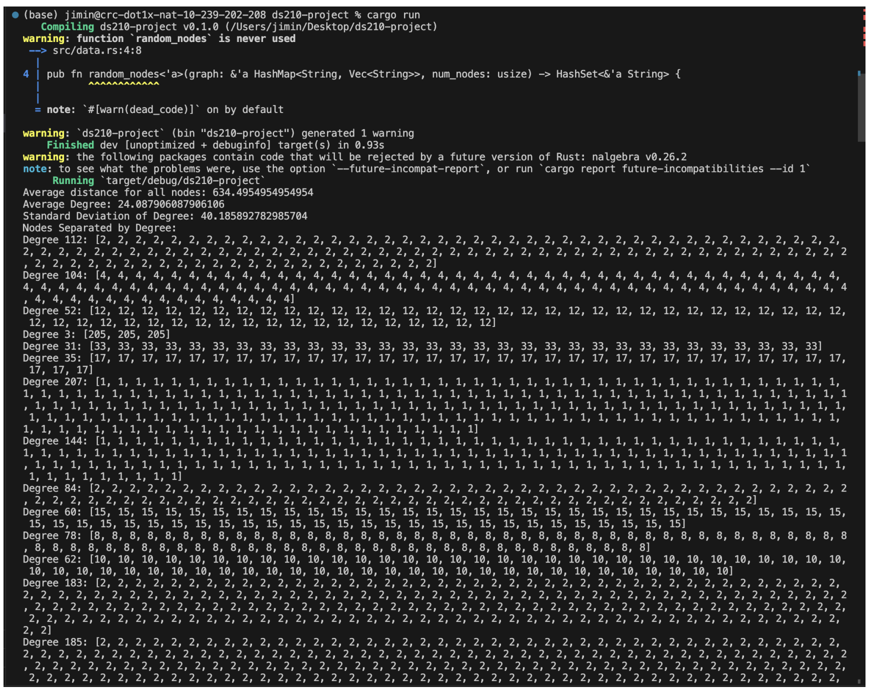
- **Degree Distribution:** The degree_distribution function calculates the degree distribution of the graph. The degree of a node is the number of its neighbors. The function returns a

HashMap where each key is a degree and its value is the number of nodes with that degree.

- **Friends of Friends Overlap:** The friends_of_friends_overlap function calculates the overlap between the set of friends of friends of the selected nodes and the set of selected nodes itself. It first creates a HashSet of all friends of friends of the selected nodes. Then it calculates the intersection of this set with the set of selected nodes and returns the size of the intersection.
- **Load Facebook Data:** The load_facebook_data function is a placeholder for a function that would load the facebook data from a given url and return a graph.

Results: explanation of output and the meaning of each result

// the code usually takes more than 15 minutes to run



1. **Average distance for all nodes: 634.4954954954954 -** This is the average shortest path length between all pairs of nodes in the graph. It's a measure of the average number of steps it takes to get from one user to another by stepping through their friends.
2. **Average Degree: 24.08790608790616 -** This is the average number of friends (i.e., edges) that each user (i.e., node) has in the graph.
3. **Standard Deviation of Degree: 40.18589278298567 -** This is the standard deviation of the number of friends that each user has. It measures how much variation there is from the average number of friends.
4. **Nodes Separated by Degree -** This is a breakdown of the number of nodes that have a certain degree. For example, "Degree 37: [24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24]" means there are 37 nodes in the graph that each have 24 friends.

Test explanation:

1. test_load_facebook_data: This test checks the load_facebook_data function. It creates a test file with a known graph structure, then calls load_facebook_data with the test file's name. It asserts that the returned graph has the correct number of nodes and that the connections between nodes are as expected.
2. test_random_nodes: This test checks the random_nodes function. It creates a small graph and calls random_nodes to select a subset of the nodes. It asserts that the returned set has the correct number of nodes.
3. test_six_degrees_of_separation: This test checks the six_degrees_of_separation function. It creates a small graph and a set of selected nodes, then calls six_degrees_of_separation with the graph and the selected nodes. It asserts that the returned average distance is correct.
4. test_average_and_std_dev_of_highest_degree: This test checks the average_and_std_dev_of_highest_degree function. It creates a small graph, then calls average_and_std_dev_of_highest_degree with the graph. It asserts that the returned average degree and standard deviation are correct.
5. test_nodes_separated_by_degree: This test checks the nodes_separated_by_degree function. It creates a small graph, then calls nodes_separated_by_degree with the graph. It asserts that the returned separation counts are correct for each degree.