A. Project Overview

Goal: This project aims to understand how people are connected within a social network by analyzing social circles data on Facebook as a graph. Each user is represented as a node, and friendships are represented as edges between them. Using basic graph theory techniques, the project examines the structure and patterns of this network. It measures how connected the network is overall by calculating the average shortest path between users, analyzes how popular or connected users are through degree distribution, and identifies which users have the most similar friend groups using Jaccard similarity. The project gives insights into the overall connectivity, clustering, and social structure of the Facebook graph.

Dataset: Social circles: Facebook

- **Source:** https://snap.stanford.edu/data/ego-Facebook.html - Stanford University
- **Size:** 4,039 nodes and 88,234 undirected edges
- **Format:** Edge list where each line is a pair of connected user IDs

B. Data Processing

How it was loaded into Rust: I used BufReader to read file line-by-line, parsed each line into two node IDs, and finally nserted each edge into an adjacency list representation using a HashMap<usize, Vec<usize>>. For some cleaning I trimmed whitespace and parsed strings into usize then constructed an undirected graph (bidirectional edges)

C. Code Structure

- **main.rs**: coordinates loading, analysis, and printing
- **data_loader.rs**: loads edge list into an adjacency list graph
- **network.rs**: defines the Graph struct and basic operations
- **analysis.rs**: contains core logic for BFS, degree distribution, and similarity measures

In network.rs (graph structs): stores an undirected graph via an adjacency list

- **Methods:**
- add_edge(u, v): Adds undirected edge
- degree(node): Returns number of neighbors
- num_nodes(): Returns total nodes

load_graph_from_file(path) (in data_loader.rs)**:** loads edge list from file and builds a graph

- Inputs file path (string) and outputs graph object. It reads file line-by-line, split each into node pairs, and add edges

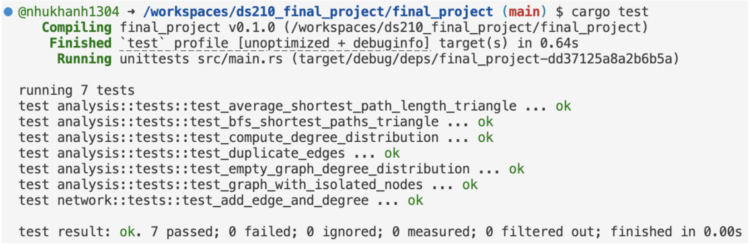
bfs_shortest_paths(graph, start) (in analysis.rs): runs BFS to find shortest path from start to every other node. It uses VecDeque and HashSet for queue and visited
average_shortest_path_length(graph, start): averages all reachable shortest paths from start. It filters out unreachable nodes and divides total distance by count
compute_degree_distribution(graph): count how many nodes have each degree. It output HashMap<degree, count>
print_degree_distribution(graph): it prints ASCII histogram of degree distribution
find_top_jaccard_similarities(graph, node, k): prints top k nodes with highest similarity to given node. For each other node, I computes Jaccard similarity (intersection / union of neighbor sets).
find_most_similar_pair(graph): does exhaustive pairwise similarity scan to find most similar node pair

Main Workflow: Firstly, main.rs calls load_graph_from_file(). Then, it also call analysis functions: average_shortest_path_length, print_degree_distribution, find_top_jaccard_similarities, and find_most_similar_pair. Laslt, it prints results to the terminal for visualization

D. Tests

test_compute_degree_distribution(). It checks graph with 3 nodes and 3 edges has correct degree counts to verify that graph parsing and edge-counting works proerly

Test Output:



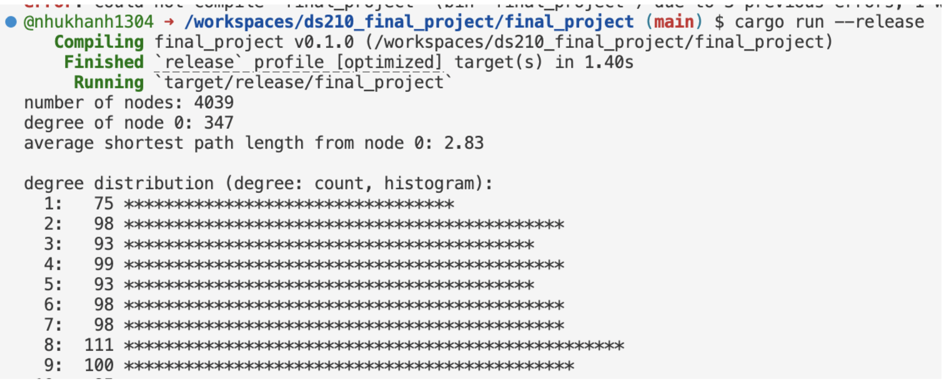
```
● @nhukhanh1304 → /workspaces/ds210_final_project/final_project (main) $ cargo test
    Compiling final_project v0.1.0 (/workspaces/ds210_final_project/final_project)
     Finished `test` profile [unoptimized + debuginfo] target(s) in 0.64s
      Running unittests src/main.rs (target/debug/deps/final_project-dd37125a8a2b6b5a)

running 7 tests
test analysis::tests::test_average_shortest_path_length_triangle ... ok
test analysis::tests::test_bfs_shortest_paths_triangle ... ok
test analysis::tests::test_compute_degree_distribution ... ok
test analysis::tests::test_duplicate_edges ... ok
test analysis::tests::test_empty_graph_degree_distribution ... ok
test analysis::tests::test_graph_with_isolated_nodes ... ok
test network::tests::test_add_edge_and_degree ... ok

test result: ok. 7 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

E. Results

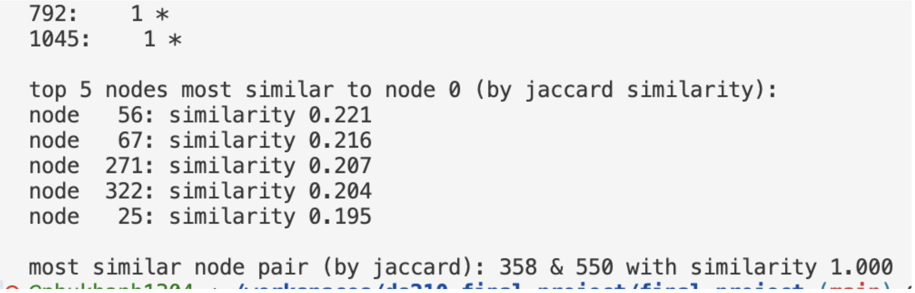
Output Summary (From cargo run --release)



```
● @nhukhanh1304 → /workspaces/ds210_final_project/final_project (main) $ cargo run --release
    Compiling final_project v0.1.0 (/workspaces/ds210_final_project/final_project)
     Finished `release` profile [optimized] target(s) in 1.40s
      Running `target/release/final_project`
  number of nodes: 4039
  degree of node 0: 347
  average shortest path length from node 0: 2.83

  degree distribution (degree: count, histogram):
    1:   75 *********************************
    2:   98 ********************************************
    3:   93 *****************************************
    4:   99 ********************************************
    5:   93 *****************************************
    6:   98 ********************************************
    7:   98 ********************************************
    8:  111 **************************************************
    9:  100 *********************************************
```

skip middle lines…..



```
  792:    1 *
  1045:    1 *

  top 5 nodes most similar to node 0 (by jaccard similarity):
  node   56: similarity 0.221
  node   67: similarity 0.216
  node  271: similarity 0.207
  node  322: similarity 0.204
  node   25: similarity 0.195

  most similar node pair (by jaccard): 358 & 550 with similarity 1.000
```

Interpretation:

The graph shows short path lengths, & some pairs even have extremely high similarity (shared neighbors). This means most FB users are separated by only a few degrees, with an average shortest path length of about 2.83 from a central user. The degree distribution shows that although most users have a moderate number of connections, a small group has a large number of connections, which suggests a hub structure. High Jaccard similarity scores between certain users shows clusters with very similar neighbor sets, highlighting the presence of tight-knit friend groups and overlapped communities.

F. Usage Instructions: Build: cargo build --release; **Run:** cargo run --release

Expected Runtime: ~15s for cargo run –release. **Input file** is facebook_combined.txt in root directory.