Dataset:

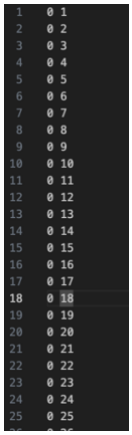
Link: https://snap.stanford.edu/data/ego-Facebook.html

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

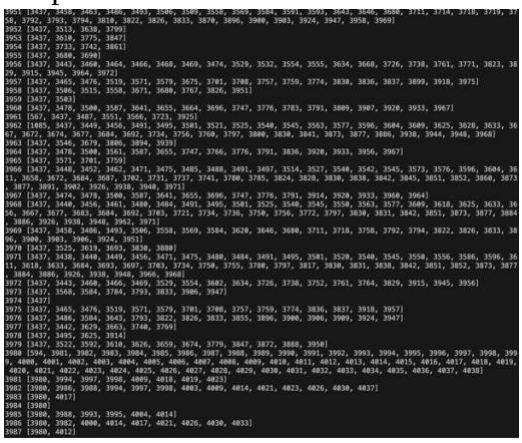
Description:

This dataset consists of friends lists from Facebook. Where each node is a person and people are connected by an edge if they are friends on Facebook. I specifically used ‘facebook_combined.txt’.

Raw data:



Graph:



This is a small part of the graph.

Code explanation:

Main.rs:
Imported modules using ‘mod’, allowing access to functions defined in separate files.
Main Function:
1. File Reading and Graph Initialization:
It attempts to read edges from the dataset file "facebook_combined.txt" using ‘read_edges_from_file’ function from ‘file_utils.rs’. It then computes the maximum vertex index based on the edges and initializes an undirected graph using the ‘Graph’ struct from ‘graph.rs’. It randomly selects a starting vertex for analysis.
2. Graph Analysis:
It does breadth-first search (‘bfs_distances’) to compute distances from the randomly selected vertex to other vertices in the graph. It then prints information about the graph: average distance between vertices, the sum of friends for each vertex, the number of mutual friends for a specific vertex, and the most/least connected nodes in the graph.
Tests:
Contains multiple tests ensuring the correctness of graph-related functions (‘bfs_distances’, ‘average_friends’, ‘calculate_mutual_friends’, ‘most_connected_node’, ‘least_connected_node’). All the tests pass.

Graph.rs:

-Graph Creation (‘create_undirected’)
The ‘Graph::create_undirected’ function initializes an empty graph with a specified number of vertices (‘n’). It iterates through the list of edges provided and adds each edge to the graph by invoking the ‘add_edge’ function. ‘add_edge’ function adds edges to the adjacency lists for both vertices in an undirected manner.
3. BFS Distances Calculation (‘bfs_distances’)
Uses Breadth-First Search (BFS) algorithm to calculate distances from a starting vertex. It initializes a distance vector with ‘None’ values for each vertex. Sets the distance of the starting vertex to ‘0’. Finally, it utilizes a queue (`VecDeque`) to perform BFS:

- Iterates through the vertices adjacent to the current vertex.
- Updates their distances if not already visited (distance is `None`) and enqueues them.
- Continues until the queue is empty, updating distances as it traverses the graph.

-Average Number of Friends (‘average_friends’)
Computes the average number of friends per person in the graph. Iterates through each vertex's adjacency list and accumulates the total number of connections (edges). Divides the total count by twice the number of vertices (‘2*num_nodes’) to consider undirected edges.
5. Mutual Friends Triangle calculation (‘calculate_mutual_friends’)
Counts the number of mutual friends for a given vertex. Considers each friend of the vertex (`friend`):

- -Iterates through their friends (`friend_of_friend`).
- Checks if ‘friend_of_friend’ is not the vertex itself and if it exists in the original vertex's friends.
- Increments the count of mutual friends accordingly.
- Divides the final count by 2 to account for double-counting in undirected graphs.

-Most and Least Connected Nodes

- Most Connected Node (‘most_connected_node’)

Finds the node with the highest degree (most friends) by using ‘max_by_key’ on the length of adjacency lists.

- Least Connected Node (‘least_connected_node’)

Finds the node with the lowest degree (fewest friends) by using ‘min_by_key’ on the length of adjacency lists.

Sure, here are write-ups for the remaining functionalities in your Rust program:

-Mutual Friends Triangle Calculation (‘calculate_mutual_friends’)

The function determines the count of mutual friends for a specified vertex within the graph:

- Iterates through each friend of the provided vertex.
- For each friend it explores their connections and checks:
    - If ‘friend_of_friend’ is not the original vertex and exists in the initial vertex's friend list.
    - Adds count of mutual friends based on the above conditions.
    - Finally, the total count of mutual friends is divided by 2 to adjust for the double-counting of undirected graphs.

-Graph Density Calculation (‘graph_density’)

The `graph_density` function computes the density of the graph based on the number of edges and vertices:

- Calculates the maximum possible edges in an undirected graph with ‘num_nodes’ vertices using the formula: (num_nodes * (num_nodes - 1)) / 2.
- Then, it divides the actual number of edges (‘edges_count’) by the maximum possible edges to determine the density.

-Graph Connectivity Check (‘is_connected’)

The function checks whether the graph is fully connected:

- Iterates through each node in the graph.
- For each node, it uses the ‘bfs_distance’ function to calculate distances from that node to all other vertices.
- If a vertex is unreachable (distance is `None`) after the BFS, the function concludes that the graph is not connected.
- Otherwise, if all vertices are reachable from each other, the graph is considered connected.

These functionalities collectively provide insights into the relationships and structure within the graph, facilitating analysis of connections, density, and connectivity among vertices.

Random Utilities Module (‘rand_utils.rs’)

Generate_random_start_vertex:

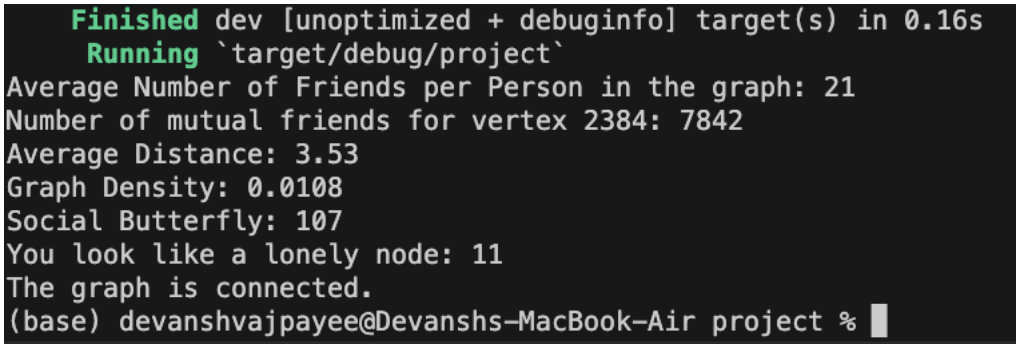
- Provides a random starting point (vertex) within the graph.
- rand::thread_rng(): Initializes a random number generator using the thread-local source of randomness.

- gen_range(0..max_vertex): Utilizes the random number generator to generate a random number within the specified range [0, max_vertex).
- max_vertex represents the upper limit for the random vertex generation. It ensures that the generated vertex is within the range of the available vertices in the graph.

File Utilities Module ('file_utils.rs')

Reading Edges from File
The 'read_edges_from_file' function in 'file_utils.rs' reads lines from a file. For each line, it splits the line by whitespace, parses the two vertex values, and stores them as edges in a vector.

Output:



Network statistics:
- Total Nodes: 4039
- Total Edges: 88234
- Average Friends per Person(Edges per Node): 21

Graph analysis:

- Average Distance Between Individuals: **3.53**. This indicates the typical path length or number of connections required to navigate from one random person to another. This metric is called "six degrees of separation", it showcases the network's overall interconnectedness. **It takes ~4 friendships to traverse from any person to another within this network.**

- Mutual Friends: The calculation of mutual friends for a specific random vertex, such as the 253 mutual friends identified for vertex 1010, identifies a significant aspect of the graph. This metric represents instances where two individuals connected to a common friend.

- Node 107 emerges as a 'Social Butterfly,' representing a highly connected and influential individual within the network.

- Node 11 appears as a 'lonely node,' indicating a comparatively isolated individual within the network structure.

- A graph density of 0.0108 indicates that the actual number of edges in the graph represents approximately 1.08% of the total possible edges that could exist in a complete graph with the same number of vertices. This level of density suggests a relatively sparse graph.

- Data validation: Checks whether graph is connected.

Within a network of approximately 4000 individuals, consistently high mutual friend counts across various vertices signify a network structure that's relatively open and interconnected.

Resources:
https://doc.rust-lang.org/stable/book/
https://gist.github.com/vTurbine/16fbb99225ad4c0ac80b24855dd61a7c
https://stackoverflow.com/questions/71189961/bfs-algorithm-tracking-the-path
https://docs.rs/graphrs/latest/graphrs/
https://www.confessionsofadataguy.com/exploring-graphs-in-rust-yikes/