

Ahmed Haidar

12/15/24

Project Writeup

Project Overview

My project aims to analyze large graph datasets in Rust by inputting a YouTube social network dataset, [com-youtube.ungraph.txt](#), with user connections represented as edges in graphs. The dataset is an undirected graph representing a social network of YouTube users, where each edge denotes a mutual connection between two users. With over a million nodes and nearly three million edges, it offers a comprehensive snapshot of user relationships. The work implemented an adjacency list-based graph data structure and provided functions for graph property analyses such as degree centrality, connected components, and average shortest path. The main program can load the dataset, clean the data, and perform various analyses. These analyses provide insight into the structure or connectivity of the graph and indicate important nodes for high overall connectivity. In each of these implementations, the main focus was on efficiency and clarity; only then would my code be able to handle big data sets.

Main Functions

The main functionality of the project is implemented through the following functions:

1. **“load_graph”**: This function loads the dataset into the graph structure. During loading, it cleans the data by removing duplicate edges and self-loops to ensure the graph is correctly structured for analysis.
2. **“calculate_degree_centrality”**: This function identifies the nodes with the highest number of connections (degree centrality) by calculating the degree for each node in the

graph. It sorts nodes by degree and outputs the top 10 nodes with the highest connectivity.

3. **“find_connected_components”**: This function identifies distinct connected components in the graph. It iterates through all nodes, exploring each unvisited node’s neighbors to group them into a connected component. It outputs the total number of connected components and the size of the most significant element.
4. **“analyze_connectivity_bfs”** (located in main.rs): This function estimates the graph's average shortest path length using BFS. It randomly samples a set of nodes, performs BFS on each one, and calculates the distances to other nodes. The average shortest path length is then estimated based on the sampled distances.

How to Run It

To run this project:

1. Download the zip file from [GitHub](#) and ensure the dataset file [com-youtube.ungraph.txt](#) is in the same directory as all your project files.
2. Open a terminal and position yourself at the root of this project.
3. Run the program with “cargo run –release,” and it will load the dataset, clean the data, and calculate several graph metrics: degree centrality, connected components, and an average shortest path length estimation. The results will be printed directly to the terminal.

Output Values

The main file must be executed with "cargo run --release" as the dataset is large so performance must be optimized. The key outputs are as follows:

- **Graph Statistics:** The program loads the dataset and reports the number of nodes and edges in the graph, giving an overview of its size and complexity.
- **Degree Centrality Analysis:** The top 10 nodes with the highest degree centrality are displayed, which show the graph's most influential nodes. Each node's degree (number of direct connections) is included.
- **Connected Components Analysis:** The program identifies the number of connected components in the graph and reports the size of the largest component, demonstrating how interconnected the nodes are.
- **Shortest Path Estimation (BFS):** Using BFS sampling, the program calculates the estimated average shortest path length. This output provides insight into how closely connected the nodes are across the graph.

Below is a screenshot of an example of the output from the project:

```
● (base) ahmedhaidar@Ahmeds-MacBook-Pro project % cargo run --release
Finished `release` profile [optimized] target(s) in 0.05s
Running `target/release/project`
Loading the graph...
Graph loaded successfully!
Number of nodes: 1134890
Number of edges: 2987624

Calculating degree centrality...
Top 10 nodes by degree centrality:
Rank 1: Node 1072 with degree 28754
Rank 2: Node 363 with degree 14641
Rank 3: Node 35661 with degree 11281
Rank 4: Node 106 with degree 10461
Rank 5: Node 482709 with degree 9762
Rank 6: Node 663931 with degree 8843
Rank 7: Node 929 with degree 7917
Rank 8: Node 808 with degree 6102
Rank 9: Node 27837 with degree 5393
Rank 10: Node 108624 with degree 4899

Analyzing connected components...

Number of connected components: 1
Largest connected component size: 1134890

Analyzing connectivity...

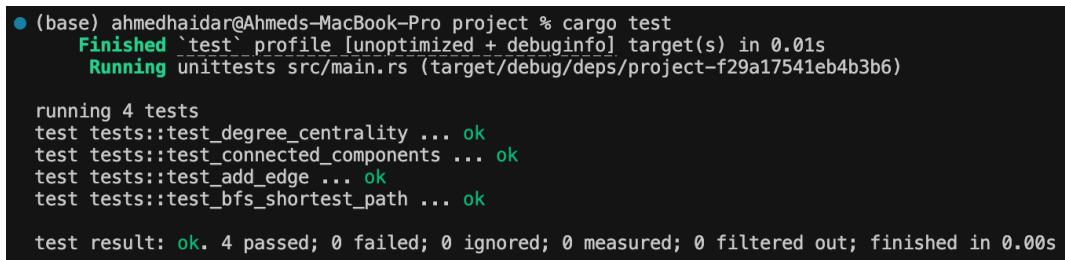
Computing average shortest path length using BFS...
Estimated average shortest path length for the graph: 5.3499
```

Test Cases

I implemented the following tests to ensure the correctness of the implemented functions:

1. **“test_add_edge”**: Verifies that adding edges to the graph correctly updates the node and edge counts.
2. **“test_degree centrality”**: Confirms that the degree centrality is calculated accurately and correctly identifies the most connected node.
3. **“test_connected_components”**: Ensures that the function correctly detects the number and size of connected components in the graph.
4. **“test_bfs_shortest_path”**: Validates the BFS implementation by testing it on a small graph and comparing the computed shortest path distances to the expected results.

Combining these functions and tests, the project analyzes the YouTube social network dataset and provides insights into its structural properties. Below is a screenshot of an example of the test output from the project:



```
(base) ahmedhaidar@Ahmeds-MacBook-Pro project % cargo test
Finished `test` profile [unoptimized + debuginfo] target(s) in 0.01s
Running unittests src/main.rs (target/debug/deps/project-f29a17541eb4b3b6)

running 4 tests
test tests::test_degree_centrality ... ok
test tests::test_connected_components ... ok
test tests::test_add_edge ... ok
test tests::test_bfs_shortest_path ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Summary & Findings

Ultimately, I successfully analyzed a large-scale YouTube social network dataset, which demonstrated the application of graph analysis techniques. Using breadth-first search, I was able to provide meaningful insights into the structure and connectivity of the graph by implementing key functions such as degree centrality, connected components detection, and average shortest path length calculation. Their inclusion of data cleaning made the analysis more accurate, while

comprehensive test cases were used to validate the correctness of the implemented functions. It was found that the YouTube social network is highly interconnected, having a single dominant connected component with most nodes and a relatively short average path length, hence reflecting the small-world phenomenon common in social networks. My primary perspective for this project was using everything we learned about graphs in Rust. However, using the skills I got in Rust from class and applying them to a real-world dataset, which I found interesting, was very intriguing.