Sun Cho U60668013

Analyzing the Wikipedia Vote Network

Goal

The goal of this project was to analyze a real-world graph dataset using Rust. The graph dataset used for this project is the Wikipedia Vote Network, which shows votes cast for promoting users to administrators on Wikipedia.

The tasks were as follows:

1. **Build the graph**: Represent the voting relationships as a graph where nodes represent Wikipedia users, and directed edges represent votes.
2. **Explore the graph:** Use Breadth-First Search(BFS) to explore the graph and count how many nodes(users) are reachable from a specific starting node.
3. **Measure connectivity:** Calculate degree centrality for each node to understand how many connections each user has.
4. **Analyze average connectivity:** Compute the average degree centrality to get an overall sense of how connected the users are in the dataset.

Dataset Source

The dataset I used is called Wikipedia Vote Network and comes from the Stanford Network Analysis Platform (SNAP) repository. The data shows votes cast in elections where Wikipedia users were nominated to become administrators.

- Source: J. Leskovec, D. Huttenlocher, J. Kleinberg. *"Signed Networks in Social Media."* CHI 2010.
- **Dataset Link**: SNAP Wiki-Vote Dataset
  - How to access: Click the link above, scroll down, download the link called Wiki-Vote.txt.gz. Then extract to get the .txt file.
- Format: A tab-separated text file (Wiki-Vote.txt), where each line contains two numbers:
  - FromNodeId: The ID of the user who voted.
  - ToNodeId: The ID of the user being voted on.
- **Data Statistics:**
  - Number of Nodes(Users): 7,115
  - Number of Edges(Votes): 103,689
  - Largest Connected Group: 7,066 users (out of 7,115 total)
  - Number of Triangles (Mutual Votes): 608,389
- The dataset provides a real-world example of a social network with relationships between users.

Code Structure

This project was organized into three main files: main.rs, graph.rs, and tests.rs. I split the program into different modules to keep the code clean and easy to understand. Below, I explain what each file does and how the code works.

Main.rs

The main.rs file is the entry point of the program. It does the following steps:

1. Read the dataset:
    a. The program reads the Wiki-Vote.txt file line by line.
    b. Lines starting with # are skipped since they are comments.
    c. Each valid line is split into two numbers, which represent a directed edge in the graph.

```
let parts: Vec<&str> = line.split_whitespace().collect();
if parts.len() == 2 {
    let from_node: u32 = parts[0].parse()?;
    let to_node: u32 = parts[1].parse()?;
    graph.add_edge(from_node, to_node);
}
```

Here the add_edge method adds the directed edge into the graph's adjacency list.

2. Build the graph:
    a. The program uses a Graph struct to store the dataset. This graph is implemented as a HashMap where each key is a node, and the value is a list of its neighbors (connected nodes).
3. Perform BFS:
    a. The program runs BFS starting from a specific node (node 30).
    b. BFS visits all nodes that can be reached from the starting node.

```
Visited Node: 30
Visited Node: 1412
BFS visited 2316 nodes in the connected component starting from Node 30.
```

4. Calculate Degree Centrality:
    a. Degree centrality measures how many connections each node has, including both incoming and outgoing edges.
5. Compute Average Degree Centrality:

a. The program calculates the average degree centrality across all nodes.

Graph.rs

The graph.rs file contains all the methods that work on the graph. The graph is stored as an adjacency list using a HashMap<u32, Vec<u32>>.

- Graph struct:

  ```
  Pub struct Graph {
      Pub adjacency_list: HashMap<32, Vec<u32>>,
  }
  ```

  The adjacency list maps each node(user) to a list of other nodes it is connected to.

- Key Methods:
  - Add_edge - adds a directed edge to the graph
  - Bfs - Performs BFS and counts how many nodes can be reached from a starting node.
  - Calculate_degree_centrality - calculates the sum of in-degree and out-degree for each node.
  - Calculate_average_centrality - computes the average degree centrality for all nodes.

Tests.rs

The tests.rs file contains tests to ensure the program works correctly. The tests include:

1. Graph Construction: Verify that edges are added correctly.
2. BFS Traversal: Ensure BFS visits the correct number of nodes.
3. Degree Centrality: Check that the degree centrality is calculated accurately.

```
#[test]
fn test_bfs_visits_correct_nodes() {
   let mut graph = Graph::new();
   graph.add_edge(1, 2);
   graph.add_edge(1, 3);
   let visited_count = graph.bfs(1);
   assert_eq!(visited_count, 3);
}
```

Sample Output

When the program runs, the following results are printed:

1. BFS Traversal:
   BFS visited 2316 nodes in the connected component starting from Node 30.
2. Degree Centrality(for first 10 nodes):
   Node 7822: Degree Centrality 1
   Node 6132: Degree Centrality 2
   Node 5130: Degree Centrality 79
3. Average Degree Centrality:
   The average degree centrality across all nodes is 29.14659

Conclusion

In this project, I was able to load and process a real-world social network dataset. Represent the graph using an adjacency list, then implement BFS to explore the graph and count connected nodes. Then, measured degree centrality to determine the importance of nodes and also verified the correctness of the program with unit tests.

This project demonstrates how graph algorithms can analyze relationships in a network.