Rheona Mehta

12/14/23

DS210 Project Write Up

In my data science project, I've developed a Rust codebase with the primary objective of analyzing a graph stored in the "Epinions.txt" file. The code utilizes a Breadth-First Search (BFS) algorithm to determine the shortest path between user-specified start and end nodes within the graph. Let's delve into the project's intricacies:

1. Reading Graph from File:

The read_file function is pivotal for parsing the content of the "Epinions.txt" file. This file adheres to a specific format where the first line indicates the total number of vertices in the graph, followed by lines representing edges between vertices. The function adeptly constructs an adjacency list using a HashMap, associating each vertex with a list of its neighboring vertices.

2. BFS Algorithm:

The BFS algorithm takes center stage in a dedicated module named "bfs.rs." The bfs function, within this module, accepts the graph, start node, and end node as parameters. It meticulously performs a breadth-first traversal, aiming to discover the shortest path between the specified nodes. The function returns a vector representing the nodes in the shortest path if one exists, and otherwise, it gracefully returns None.

3. User Interaction:

The main function, residing in the entry point module "main.rs," facilitates user interaction. It prompts users to input the start and end nodes for the BFS operation, utilizing standard input/output mechanisms. The code ensures the graph is not empty and gracefully handles user input errors, persistently prompting users until valid node numbers are provided.

4. Testing:

Rigorous testing is an integral part of the project, orchestrated through the inclusion of a set of unit tests within the "tests" module. These tests cover various scenarios, including reading valid and invalid input files, successfully finding paths using BFS, and handling cases where no path exists. Leveraging the Rust testing framework, these tests contribute to the reliability and robustness of the code.

5. Running the Code:

Executing the code necessitates the presence of the "Epinions.txt" file at the specified path. Upon running the program, users are prompted to input start and end nodes, and the program outputs the shortest distance between these nodes using the BFS algorithm. The provided unit tests can be executed to validate the correctness of the code's functionality.

In the design philosophy of my Rust code, I intentionally structured it into two modules, "main.rs" and "bfs.rs," to ensure modularity and maintainability. The former serves as

the entry point, orchestrating user interaction and program flow, while the latter encapsulates the BFS algorithm, fostering code modularity and potential reuse in other projects.

Within the "bfs.rs" module, the heart of the code lies in the bfs function. This function adeptly employs data structures such as HashSet, VecDeque, and HashMap to facilitate BFS traversal, and it dynamically constructs the shortest path if one is found. The separation of modules not only enhances code readability but also provides a foundation for maintaining and evolving the BFS algorithm independently, ensuring changes do not reverberate throughout the entire codebase. This deliberate design choice reflects a commitment to creating an organized, modular, and efficient solution for discovering the shortest path in a graph.