Zhijia Yang

DS210

Final Project Writeup

April 30, 2025

Professor Leonidas Kontothanassis

Final Project Writeup

A. Project Overview

- Goal: What question are you answering?

Who are the most influential person in FB-Pages_Sport and what proportion can this person reach in three steps breath first search?

- Dataset:

https://networkrepository.com/fb-pages-sport.php

Nodes:13.9K

Edges:86.8K

B. Data Processing

- How you loaded it into Rust.

I download the edges zip file and nodes zip file from the webside directly.

- Any cleaning or transformations applied

I transfered the type of the file into .csv which was simple to read. Due to there are repeating names in the nodes file. I had to give each name a specific number after their name to make them special. Finally I collected these unique names in the node hashmap which was returned.

C. Code Structure

- Modules Purpose of each and rationale for organization.

I have two modules, a graph module, a test module. The graph module has the struct for GraphData and the implementation for GraphData. The test module has two test function one is tested for the three_steps_bfs and another one is tested for the popular person.

- Key Functions & Types (Structs, Enums, Traits, etc)

The purpose for the GraphData struct is to contain the core components of the social graph, including nodes, edges, and connections. The read_nodes function reads node data, where each record contains ID, name and new ID. However, I only use the last two parts of the record, because the orginal id is nonsense. Since names may be duplicated in the name list, this funciton uses a counter (name_counts) to track how many times each name has appeared and appends a numeric suffix to ensure name uniqueness. It returns a HashMap mapping each unique ID to its corresponding name.

Similarly, the read_edges function reads undirected edges information from edges file and returns a vector of (from, to) pairs, each representing a directed connection between two node unique IDs.

The find_connections function builds an adjacency list from the provided edges and nodes. For each node, it checks which edges start from that node and compiles a list of directly connected neighbors. The result is a HashMap where each key is a node ID, and each value is a vector of IDs that can be reached from it in one step.

The popular_person function analyzes this adjacency list to determine which person (node) has the most outgoing connections. It iterates through each node's neighbor list, identifying the node with the longest list and returning the corresponding name. This allows the program to identify the most "influential" person in the graph based on direct reach.

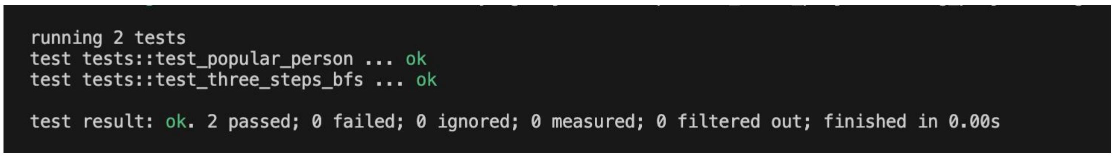
Finally, the three_steps_bfs function uses a breadth-first search to determine how many people can be reached from a given starting name within a fixed number of steps (typically three). It begins by locating the start node by name match, then performs BFS up to the specified depth, using a queue and a visited set to avoid revisiting nodes. The function returns the number of distinct nodes reached within the allowed depth, excluding the starting node. Together, these components enable analysis of both direct and indirect influence in a social network.

- Main Workflow

At a glance, the graph module provides all the core functions used in main: it reads node and edge data from CSV files, builds the graph's adjacency list, identifies the most connected (popular) person, and computes how many others that person can reach in three steps using BFS. These functions work together to load data, analyze popularity, and evaluate influence spread in the network.

D. Tests

- cargo test output :



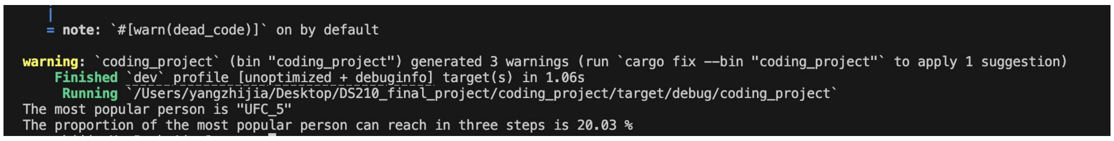
```
running 2 tests
test tests::test_popular_person ... ok
test tests::test_three_steps_bfs ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

- For each test: what it checks and why it matters.

The two tests serve to validate the key functions that drive its results. The test_three_steps_bfs test checks whether the breadth-first search algorithm correctly computes the number of unique nodes that can be reached from a given starting node within three steps. This is important because it ensures that the BFS logic is functioning properly and that the program can accurately evaluate indirect influence within the network. Additionaly, the test_popular_person test check whether the function responsible for identifying the most popular individual—defined as the person with the most outgoing connections—returns the correct result. To sum up, these tests confirm the result can be returned successful.

E. Results



```
    |
    = note: `#[warn(dead_code)]` on by default

warning: `coding_project` (bin "coding_project") generated 3 warnings (run `cargo fix --bin "coding_project"` to apply 1 suggestion)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.06s
     Running `/Users/yangzhijia/Desktop/DS210_final_project/coding_project/target/debug/coding_project`
The most popular person is "UFC_5"
The proportion of the most popular person can reach in three steps is 20.03 %
```

- Interpretation in project context :

The most popular person in this dataset is UFC_5. I believe the original name of this person is UFC. Due to the repeated name, he's the fifth person who's name is UFC. Therefore, his unique name is UFC_5. And The proportion of UFC_5 can reach in three steps breadth first search is 20.03%. It means he can reach 20.03% people in this dataset in three steps.

F. Usage Instruction

- How to build and run your code.

To build and run the code, I firstly create a folder on GitHub and connect the link with my vscode software. Then I use cargo build to create a new file and put the nodes.csv and edges.csv in to the project directory at the same time. And, I run the code through cargo run and cargo run -- release for several times.

- Description of any command-line arguments or user interaction in the terminal.

No additional command-line arguments are needed, and there is no user interaction during runtime; all inputs are file-based, and outputs are printed directly to the terminal.

- Include expected runtime especially if your project takes a long time to run.

The expected runtime is within 3 seconds. I tried cargo run --release at first so the dependencies can be built up quickly. And the rest of the time, I used cargo run to get the output, which took up almost 3 seconds.

G. AI-Assistance Disclosure and Other Citations

ChatGPT: I asked ChatGPT for Some parts. The first part is in the read_nodes function, I didn't know how to add a number after the base_name. I had the idea to create a random number, but I thought it might look messy if I did that. Therefore, I asked chatgpt and got the idea to created a count number and put it after the base_name. The second part is in the three_steps_bfs function, I had to change the name of the start person into the id of it. However, I didn't want to add a loop to find out the name. Therefore, I asked chat and got a.find() function which was helpful for me.