DS210 Project - Paulo Obnial

Disclaimer: This project has a pretty long runtime, so please use cargo run –release

Project Background + Findings:

I found this dataset through Konect and decided to use this airport and routes dataset as it was similar to a euroroads dataset that was recommended to me by a TA. In this project, I wanted to find out which airports/service centers (notes) were most significant in the network when creating a preferred flight route (edges). I used two types of centrality analysis', degree centrality and betweenness centrality algorithms to find the most significant airports/service centers in the dataset. When I downloaded the .tsv file, I converted the file into .csv and added labels "from" and "to" to signify the directed nature of the dataset, which also made it easier to clarify my source and target in my AirportRoute structure. I chose to do a degree centrality analysis as it was something that I knew how to do and felt like that it would offer me a basic idea of the nodes with the most out degree connections. I also decided to do a betweenness centrality analysis that counted how many times a node showed up as an intermediate node when finding ths shortest path from one node to another. In my findings, I found that in both algorithms, the top two nodes, 68 and 52, were ranked #1 and #2 respectively. When it came to the next three ranked nodes, each analysis had unique nodes and they didn't overlap. This shows that outgoing connections doesn't necessarily correlate to being a frequent node on the shortest path. In the project, the part I had the most trouble with was my betweenness centrality. I would say that

ChatGPT helped me a lot with understanding and creating the logic, especially with the fact that I had such large numbers coming in the output. The betweenness centrality algorithm isn't normalized which is why the value is so high. In addition, the runtime of the algorithm takes a long time as I simplified the betweenness algorithm so that it could easily understandable to me, but I made it computationally expensive for the sake of clarity. Outside of using a test case to verify the reliability of the betweenness output, I also used a python variant that I programmed using Colab so that it could also calculate its own betweenness output. As shown (see below), both betweenness centralities had the same top two scores, with similar differences in scores, while the next third and fifth were different, but the fourth highest was the same rank. Between both the Python and Rust betweennness outputs, they shared 4 of the same nodes in the top 5 rankings. Overall, I felt like I was able to solidify both nodes 68 and 52, as the most important aiports in the system respectively when creating a preferred route.

Code Explanation:

Read_csv.rs:

```rust
pub fn read_file(file_path: &str) -> Result<Vec<AirportRoute>, Box<dyn Error>> {
    let file: File = File::open(file_path)?;
    let mut csv_reader: Reader<File> = ReaderBuilder::new() ReaderBuilder
        .has_headers(yes: true) &mut ReaderBuilder
        .trim(Trim::All) &mut ReaderBuilder
        .from_reader(rdr: file);

    let mut routes: Vec<AirportRoute> = Vec::new();

    for result: Result<AirportRoute, Error> in csv_reader.deserialize() {
        let route: AirportRoute = result?;
        routes.push(route);
    }
```

The read_file function reads a CSV file and parses its contents into a vector of AirportRoute structs and accepts the file (airport.csv) as a string slice (`&str`).
The function returns a a vector of AirportRoute structs. I used ChatGPT a bit to help me get some of the necessary libraries that were necessary to clean the data and make it usable.

graph_construction.rs:

The `Graph` Struct

- Central to this code is the `Graph` struct, which encapsulates the graph's data structure.
- The struct uses a `HashMap` to represent the adjacency list, a common data structure for graphs. Each key in the `HashMap` is a `u32` representing a node (airport) ID, and the associated value is a `Vec<u32>`, a list of node IDs to which there is a directed edge from the key node.
- This design choice effectively models a directed graph, where each node can have multiple outbound connections to other nodes.

- The `new` function provides a way to create an instance of the `Graph` struct. It initializes the `Graph` with an empty `HashMap`, signifying an empty graph at the start.
- The `add_node` method allows for adding a new node to the graph. It checks if the node (airport) already exists in the adjacency list and, if not, adds it with an empty list of

edges. This method ensures that each node in the graph is unique and prepared to have edges added to it.

- The `add_directed_edge` method is used to create a directed edge from one node to another. It adds an entry in the originating node's (`from`) list of edges, pointing to the destination node (`to`). This method is pivotal in constructing the connectivity of the graph, allowing the representation of routes from one airport to another.

graph_analysis.rs:

Main.rs:

Output:

In addition, the code prints each of the nodes and their score for both the degree and betweenness centrality, but I didn't really focus on it for my writeup.

Output of Betweenness Centrality on Python:

```
[(68, 209203.5087123463),
 (52, 105275.45783682672),
 (212, 96229.26094734584),
 (312, 89124.50303031302),
 (135, 85309.3382324847)]
```

Sources:

http://konect.cc/networks/maayan-faa/

https://axuplatform.medium.com/centrality-analysis-832ae76a6eca

https://qiskit.org/ecosystem/rustworkx/dev/tutorial/betweenness_centrality.html

ChatGPT

https://lib.rs/crates/graphrs