DS210 Final Project Write-up

Analysis of US Highway Network

Name: Hongyao Shao

Dataset:

https://www.kaggle.com/datasets/webdevbadger/us-highway-streets?resource=download

"US Highway Streets"

For this project, I plan to use the dataset *US Highway Streets*, Data produced in the collaborative workspace for data scientists, Kaggle. It includes data that presents the highway system of the United States. This dataset is interesting to me because I particularly enjoy road trips; at the same time, the US has one of the most extensive and complex transportation networks in the world. From my own experience, I have had trouble finding the right path on highways, it's still very complicated when many interstate highway merges. Analyzing this network can provide insights into the efficiency of transportation systems and help identify areas for improvement.

Problem Statement:

The primary question I aim to address is: Which state or areas have the best average distances, which could indicate the prosperity of that area. Also, how can the efficiency and connectivity of the United State's transportation network be optimized to enhance commuter experience and

reduce travel time? I plan to use the degree centrality, and closeness centrality to investigate the network's structure, identify critical nodes and edges.

Writeup

I did the Data Collection and Preprocessing. The dataset includes the following information:

street_number: The street number of the highway. Format is US #, i.e. US 1.
street_name: The street name of the highway.
states: The states associated with the highway.
formed: The year the highway was formed. yyyy format.
removed: The year the highway was removed. yyyy format.
length_(mi): The length of the highway in mile.
length_(km): The length of the highway in km.
southern_or_western_terminus: The southern or western terminus.
northern_or_eastern_terminus: The northern or eastern terminus.
wiki_link: The link to the highway's Wikipedia page.
wiki_id: The ID associated with the highway's Wikipedia page.
coordinates_name: The folder name containing the associated coordinates.
icon_name: The plate image file name associated with the highway.
notes: Notes about the highway.

In the code, I designed a Highway **struct** in Rust to manage and represent the details of a highway. This struct includes various fields:

street_name: A String to store the name of the highway.
states: A String that lists all the states the highway passes through.
formed: An Option<i64> which holds the year the highway was formed, if available.
removed: An Option<i64> for the year the highway was decommissioned, if applicable.
length_mi: A floating-point number (f64) that represents the length of the highway in miles.
southern_or_western_terminus: A String describing the starting point of the highway in the south or west.
northern_or_eastern_terminus: A String that describes the ending point of the highway in the north or east.

I also implemented the describe method for the Highway struct, which provides a string representation of the highway's details. This method uses the format! macro to compile a

formatted string that includes all the relevant information about the highway. To account for the possibility that the formed and removed years might not be present (hence the Option type), I used map_or with a default value of "N/A". This method ensures that even if some details are missing, the string representation still includes placeholders, avoiding any run-time errors due to unwrapping None values.

This design allows for easy and safe management of highway data, providing both flexibility (with optional values) and detailed, human-readable descriptions of each highway instance.

Then, I created a reader as well as filter in the mod `data_prep`, through this module, I read the data from the csv file and also remove those highways that had been removed.

Function: `fn build_graph(highways: Vec<Highway>) -> UnGraph<String, ()>` takes a vector of Highway structs as input and builds an undirected graph where each node represents a state, and edges represent highways connecting these states. Here's a step-by-step explanation of how I implemented this:

Graph and Node Indices Initialization:

I created an undirected graph, graph, where each node is identified by a String representing the state, and edges don't have weights. I also initialized a HashMap called node_indices to store the mapping of state names to their respective node indices in the graph.

Processing Each Highway:

For each highway in the input vector, I extracted the string containing the list of states it passes through (highway.states). I cleaned and split this string into a vector of state names (states_vec).

This involved removing surrounding brackets, trimming spaces, and splitting the string at commas.

Adding Nodes to the Graph:

I iterated over the states_vec. For each state: If the state was not already added to graph, I added it as a new node using graph.add_node and stored its index in node_indices.

Connecting States with Edges:

While iterating through the states_vec, for each state after the first (i > 0), I retrieved the node indices of the current state and the previous state from node_indices. I then added an edge between these two states in the graph using graph.add_edge. This ensures that all states listed for a highway are connected in sequence.

Returning the Graph:

After processing all highways and constructing the graph with all necessary nodes and edges, the graph is returned.

Mod: `mod calculate_centrality`

To make my project more clear, I put my different type of calculation of centrality in to a single module.

Degree Centrality:

The function degree_centrality calculates the degree centrality for each node (state) in the graph. This is achieved by iterating over each node in the graph and counting the number of edges connected to it (i.e., its neighbors). The results are collected into a HashMap where each state is mapped to its degree centrality score.

Closeness Centrality:

The function closeness_centrality computes the closeness centrality for each node. Closeness centrality is typically calculated as the reciprocal of the sum of the shortest path distances from a node to all other nodes in the graph. I utilized Dijkstra's algorithm to find the shortest path lengths from each node to all other nodes. This is done using the dijkstra function from the petgraph crate, which efficiently finds the shortest paths from a source node to all other nodes.
For each node, if the total distance to all other nodes is greater than zero, the closeness centrality is calculated as 1.0 / total_distance. If the total distance is zero (which technically shouldn't happen since it would indicate no connections), the centrality is set to zero.

Main():

In my final step, I put all my statements together, and my output is a hashmap of the degree centrality of all states and a hashmap of the closeness centrality of all states.

Tests:

I made three tests for my important functions, and all of them run a success.

Results:

Top three states by degree centrality:

North Carolina: 49

Virginia: 46

Oklahoma: 45

Top three states by closeness centrality:

Missouri: 0.007142857142857143

Tennessee: 0.006896551724137931

Iowa: 0.00684931506849315

Degree centrality measures the number of direct connections (edges) a node has in the graph. In the context of your highway network:

With a degree centrality of 49, North Carolina has the highesconnections to other states. This suggests that North Carolina has direct highway links to 49 other states, indicating a significant role as a transportation hub or a centrally located state within the network.and the following are Virginia and Oklahoma (46 and 45)

Closeness centrality measures how close a node is to all other nodes in the network, calculated by the inverse of the sum of the shortest paths to all other nodes. A higher closeness centrality indicates that a node can reach all other nodes in the network more quickly (on average), which in the context of states connected by highways suggests efficiency in travel and transport times:

Missouri has the highest centrality of 0.007142857142857143 Its high score indicates that, on average, it has the shortest paths to all other nodes (states) in the network. Missouri's geographical position as centrally located could be contributing to this, making it a strategic state for logistics and travel across the network. And the following states are Tennessee (0.006896551724137931) and Iowa (0.00684931506849315).

Summary

The centrality measures highlight the strategic importance of these states in the highway network. The high degree centrality states are pivotal as major junctions or links within the network, whereas the high closeness centrality states facilitate quick and efficient travel across the broader network, positioning them as key nodes in the overall structure of national transportation and logistics. From the result, we can conclude that states like Missouri,

Tennessee, and Iowa, with high closeness centrality, are ideally positioned for logistics hubs. Companies might prefer these locations for distribution centers because products can be transported quickly and efficiently across the country. Also, High degree and closeness centralities can attract businesses that benefit from reduced transportation costs and improved access to markets, such as manufacturing and retail. In conclusion, these states seem to be very important for the whole country because they make its life active and dynamic. Besides, centrality metrics indicate that those states can be called functional as all national mobility and logistics greatly depend on them. It means that if a state is located this way it should understand its significance and do everything possible not only to develop but also maintain appropriate level of logistic services which will definitely improve life quality not only inside but also outside the state itself.

Citations:

https://chatgpt.com/

https://www.kaggle.com/datasets/webdevbadger/us-highway-streets?resource=download