```
Shortest path from 1182134 to 1182134 takes 0 steps.
Shortest path from 1515562 to 1515562 takes 0 steps.
Shortest path from 1515562 to 1515550 takes 1 steps.
Shortest path from 936725 to 936542 takes 1 steps.
Shortest path from 936725 to 936725 takes 0 steps.
Shortest path from 936725 to 936541 takes 2 steps.
Shortest path from 1258235 to 1258235 takes 0 steps.
Shortest path from 1258235 to 1273328 takes 1 steps.
Shortest path from 1070216 to 1070173 takes 1 steps.
Shortest path from 1070216 to 1070216 takes 0 steps.
Shortest path from 1941264 to 1941295 takes 1 steps.
Shortest path from 1941264 to 1941264 takes 0 steps.
Shortest path from 801412 to 801412 takes 0 steps.
Shortest path from 801412 to 782217 takes 1 steps.
Shortest path from 135368 to 135368 takes 0 steps.
Shortest path from 135368 to 135366 takes 1 steps.
Shortest path from 591973 to 591973 takes 0 steps.
```

```
Shortest path from 971397 to 971397 takes 0 steps.
Shortest path from 971397 to 971371 takes 1 steps.
Shortest path from 411141 to 411141 takes 0 steps.
Shortest path from 411141 to 411142 takes 1 steps.
Shortest path from 843341 to 843026 takes 1 steps.
Shortest path from 843341 to 843341 takes 0 steps.
Min: Some((70, 71))
Max: Some((1970885, 1971229))
Mean: (10000, 987100.3389)
PS C:\Users\Daniel\Documents\210 Rust HW\Project\project>
```

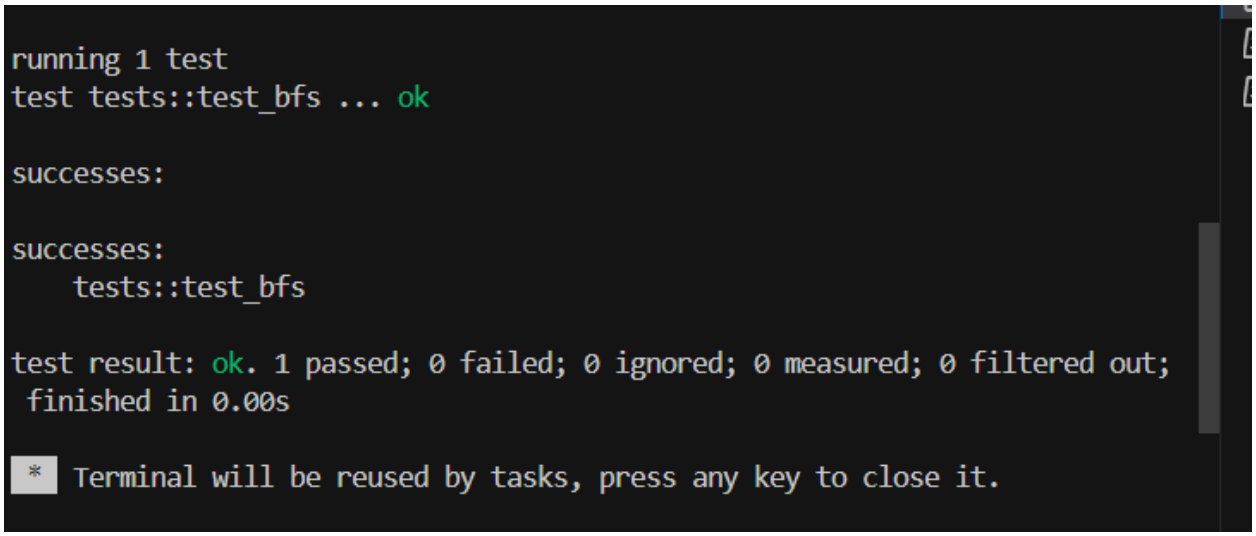
For my project, I created a Road network analysis of California. I was trying to manipulate data structures effectively, utilize external libraries for random sampling, and implement algorithms for graph traversal and basic statistical computations.

The main function starts by reading a set of edges from a file, explicitly choosing a random subset of 10,000 lines from the file located at a specified path. This is achieved through the read_file function, which employs reservoir sampling to ensure a random selection of lines without loading the entire file into memory.

After acquiring the edges, they are converted into an adjacency list using the adj_list function. This adjacency list represents the graph, where each node maps to a list of nodes to which it is directly connected, facilitating the graph traversal operations that follow.

Using the adjacency list, a breadth-first search (BFS) algorithm is executed for each node to determine the shortest paths to other nodes within the graph. This implementation highlights the connectivity and structure of the graph and outputs the shortest paths or indicates the absence of paths between nodes.

Additionally, I implemented statistical functions to analyze the graph's structure further. These functions calculate the adjacency list's minimum, maximum, and mean values. For instance, calculate_min and calculate_max traverse the adjacency list to find the smallest and largest values. Calculate_mean computes the average number of connections per node, providing insights into the graph's density.



The project also includes a module with unit tests for the BFS functionality to ensure correctness. These tests verify the distances calculated by BFS against expected values in a controlled graph scenario. If the test vectors go to themselves, it gives a 0. If the vectors go one step up, it outputs a 1. If the test vector goes up by two steps, it outputs 2.

As a testament to comprehensive software engineering practices and a deep understanding of algorithmic and data structure manipulation in real-world applications, this project seamlessly integrates file handling, graph theory, randomness, and statistical analysis using Rust.