For my DS210 project I decided to use a dataset that shows the network of a European country's roads and try to evaluate the Six Degrees of Separation in a specific road network, exploring the typical distance between pairs of locations and analyzing the degree distribution. I chose this mainly because it replicates how emergency response teams might use to get to civilians in the shortest time possible. My vertices would be intersections and endpoints and my edges would be the roads.

In order to execute this idea I first split my project into three different modules. In my first module, the csv reader, I created a function called read_csv that reads data from a CSV file specified by the provided filename. It opens the file, creates a buffered reader for efficient line-by-line reading, and iterates through each line, where using commas as separators, trims whitespace from each field, and collects them into rows. For my second module, I defined a struct AdjacencyList representing an undirected graph using an adjacency list. The struct uses a HashMap, where each vertex is associated with a vector of its adjacent vertices. The from_csv_data method is implemented to construct an AdjacencyList from CSV data. For each edge in the CSV, the method adds forward and reverse edges to the graph to account for the undirected nature of the graph. The accompanying test, test_from_csv_data, validates that the adjacency list is constructed correctly from a sample CSV dataset, checking the graph's size and specific relationships between vertices and their neighbors. For my last module I created a function, bfs, that implements the breadth-first search algorithm on an undirected graph represented by an adjacency list (graph). It starts from a given start vertex and explores neighboring vertices level by level until it reaches the end vertex. The function uses a HashSet (visited) to keep track of visited vertices, a VecDeque (queue) to manage the order of exploration, and a HashMap (parents) to store the parent vertices for reconstructing the shortest

path. The algorithm iteratively double-ended queues vertices, marks them as visited, checks for the destination vertex, and enqueues unvisited neighbors with incremented distances. If a path is found, the function returns the distance; otherwise, it returns None. The provided code uses String types for vertex labels, and the cloning operations ensure proper ownership and lifetime management. In my main function I also utilize each of the functions I created to find what percentage of shortest taken only used 6 degrees.

The outputs produced a range of 1 degrees all the way to 62 degrees. My results indicated that only 1.63% of paths taken between two vertices had a degree of 6. The results show that only 1.63% took going through 6 intersections to get to their destination. This makes sense due to my dataset having more than a 1000 nodes meaning there will be a higher percentage needing to go use multiple roads and intersections to eventually reach a destination. For example, trying to get to a destination that is on the other side of the country would take more than 6 degrees (intersections). Around 3.57% shortest paths were made up of 14 vertices which was the highest percentage and as the number of degrees (intersections) increased less were using these numbers to get to their destination which lines up because we are trying to find the shortest path.

***Note*: When you run my project it will say running and take a very long time, however I know it runs from my outputs being displayed below.**

```
Percentage of pairs with 38 degrees: 0.54%
Percentage of pairs with 51 degrees: 0.04%
Percentage of pairs with 29 degrees: 1.26%
Percentage of pairs with 41 degrees: 0.35%
Percentage of pairs with 5 degrees: 1.43%
Percentage of pairs with 22 degrees: 2.56%
Percentage of pairs with 16 degrees: 3.37%
Percentage of pairs with 1 degrees: 0.21%
Percentage of pairs with 49 degrees: 0.06%
Percentage of pairs with 3 degrees: 0.67%
Percentage of pairs with 48 degrees: 0.09%
Percentage of pairs with 13 degrees: 3.64%
Percentage of pairs with 43 degrees: 0.25%
Percentage of pairs with 34 degrees: 0.88%
Percentage of pairs with 31 degrees: 1.11%
Percentage of pairs with 46 degrees: 0.14%
Percentage of pairs with 37 degrees: 0.62%
```

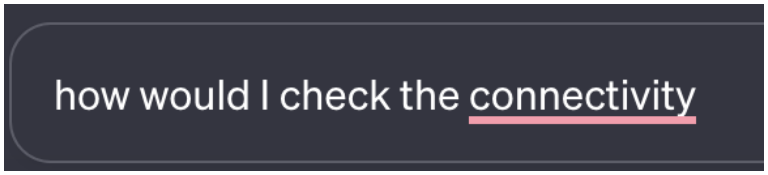
```
Percentage of pairs with 35 degrees: 0.79%
Percentage of pairs with 40 degrees: 0.40%
Percentage of pairs with 52 degrees: 0.03%
Percentage of pairs with 25 degrees: 1.92%
Percentage of pairs with 21 degrees: 2.74%
Percentage of pairs with 33 degrees: 0.96%
Percentage of pairs with 7 degrees: 2.24%
Percentage of pairs with 53 degrees: 0.02%
Percentage of pairs with 55 degrees: 0.01%
Percentage of pairs with 30 degrees: 1.18%
Percentage of pairs with 10 degrees: 3.36%
Percentage of pairs with 26 degrees: 1.71%
Percentage of pairs with 62 degrees: 0.00%
Percentage of pairs with 54 degrees: 0.01%
Percentage of pairs with 24 degrees: 2.15%
Percentage of pairs with 28 degrees: 1.38%
```

```
Percentage of pairs with 44 degrees: 0.21%
Percentage of pairs with 50 degrees: 0.05%
Percentage of pairs with 14 degrees: 3.57%
Percentage of pairs with 32 degrees: 1.03%
Percentage of pairs with 9 degrees: 3.03%
Percentage of pairs with 2 degrees: 0.39%
Percentage of pairs with 20 degrees: 2.89%
Percentage of pairs with 59 degrees: 0.00%
Percentage of pairs with 39 degrees: 0.47%
Percentage of pairs with 27 degrees: 1.53%
Percentage of pairs with 36 degrees: 0.70%
Percentage of pairs with 11 degrees: 3.55%
Percentage of pairs with 8 degrees: 2.63%
Percentage of pairs with 47 degrees: 0.11%
Percentage of pairs with 57 degrees: 0.00%
Percentage of pairs with 58 degrees: 0.00%
Percentage of pairs with 19 degrees: 3.02%
```

```
Percentage of pairs with 4 degrees: 1.03%
Percentage of pairs with 18 degrees: 3.14%
Percentage of pairs with 6 degrees: 1.83%
Percentage of pairs with 56 degrees: 0.01%
Percentage of pairs with 12 degrees: 3.65%
Percentage of pairs with 42 degrees: 0.30%
Percentage of pairs with 60 degrees: 0.00%
Percentage of pairs with 15 degrees: 3.47%
Percentage of pairs with 17 degrees: 3.25%
Percentage of pairs with 61 degrees: 0.00%
Percentage of pairs with 23 degrees: 2.36%
Percentage of pairs with 45 degrees: 0.18%
```

Although my results did not prove that the idea that you can get to any place can be reached within 6 degrees it does lead me to question how emergency services navigate to get to their destination as fast as possible. If I could improve this project I would adjust how much I was cloning which was probably why my run time for a large data set took such a long time and also possibly sort the percentages by the number of degrees for better readability.

Resources that I used:




```rust
// Check specific connections
assert!(adjacency_list.graph.get("0").unwrap().contains(&"1".to_string(
assert!(adjacency_list.graph.get("1").unwrap().contains(&"0".to_string(
assert!(adjacency_list.graph.get("1").unwrap().contains(&"2".to_string(
assert!(adjacency_list.graph.get("2").unwrap().contains(&"1".to_string(
```