DS 210 Final Project:
Finding the most important intersections in 6 degrees of separation

What does this project do?

For this project, I have chosen a weighted dataset on the road network of San Joaquin county in California. It has around 18,000 nodes, where each node represents an intersection and each edge represents a road connecting each intersection. I wanted to find the most important nodes in the dataset by checking which node has the most reachable nodes for each degree: 6, 5, 4, 3, 2, and 1. However, I thought that sometimes a node may have more reachable nodes, say in the 2nd degree but it may also be farther away from them, i n other words, it may have a higher average distance from them. So I wanted to classify the most important node by seeing which node has the most reachable nodes and the smallest average distance (using a modified version of Dijkstra's algorithm) to all the reachable nodes in that particular degree.

This is very useful because, not only can I see which are the most important nodes based on reachable nodes and average distance, I can also check if the most important node for 1 degree is also the most important node for degree 2 and so on. My hypothesis is that the most important node for a certain degree should also be the most important node for the next degree because often counties have certain intersections that connect major parts.

Explaining each module of the project

1. **graph.rs**

The graph.rs module forms the foundation of the project by defining the core data structures and a function related to graph representation. It introduces types such as Vertex, Distance, and WeightedEdge to encapsulate the elements of a weighted graph (nodes, distances, and edges). The WeightedGraph struct is a central data structure that holds all edges of the graph and the number of vertices (n). The function create_adjacency_list takes a WeightedGraph as input and constructs an adjacency list representation of the graph. This representation is crucial for efficient traversal and analysis of the graph, as it allows for quick access to a node's immediate neighbors and the distances to them. This module essentially lays the groundwork for graph-related operations, enabling further analysis and algorithms to be applied to the data structure.

2. **Reader.rs**

The reader.rs module is essential for loading graph data from a file into the program. It processes each line of the file to extract information about the graph's edges, including start and end nodes, and their weights. This module uniquely addresses undirected graphs by adding both directional edges to the graph's data structure. It dynamically calculates the total number of vertices in the graph, ensuring an accurate representation. The result is a WeightedGraph object ready for further analysis, crucial for the project's graph-processing capabilities.

3. **Algorithm.rs**

The algorithm.rs module is the crux of the project, hosting the primary algorithms for analyzing the graph's connectivity and identifying key nodes. This module is essential due to its implementation of graph analysis algorithms and its handling of certain Rust-specific constraints.

- Implementing PartialOrd for State

In Rust, floating-point numbers (f64 in this case) don't implement the Ord trait because not all floating-point values (like NaN) have a total order. However, BinaryHeap requires elements to be ordered. To overcome this, State struct, which includes a floating-point distance, implements PartialOrd and Ord. This allows us to define a custom ordering for State objects, which is necessary for their use in a priority queue (BinaryHeap). The ordering is based on the distance, with the lowest distance having the highest priority.

I learned about how to use a State struct for ordering with AI.
I have given extensive comments for this particular code to demonstrate the order of execution.

- Function: analyze_by_degrees_of_separation

This function analyzes the graph to determine the reachability of nodes within a specified number of degrees of separation (up to max_degree). For each node in the graph, it computes the shortest distance to all other nodes within the given degrees of separation. This is achieved using a modified Dijkstra's algorithm, where each node's distance and degree of separation are tracked. The algorithm stops exploring a node's neighbors if it exceeds the specified degree of separation. The function then aggregates these results, giving the number of nodes reachable and the average distance for each degree of separation.

- Function: find_detailed_important_nodes_by_degree

This function identifies the most important nodes for each degree of separation based on their connectivity and average distance to reachable nodes. It first sorts nodes at each degree of separation by their reachability and average distance. The top 10% of nodes (or at least one node) are considered, and the node with the shortest average distance among these is selected as the most important for that degree. If the most important node has 0 reachable nodes, it's marked as None. This approach highlights the nodes that are most central or influential within various degrees of separation in the graph, offering insights into the graph's structure and key elements.

4. **main.rs**

In main.rs, the program integrates modules to analyze a graph's connectivity and identify key nodes. It starts by reading graph data, transforming it into an efficient format for analysis. The core functionality involves analyzing the graph to determine nodes' reachability within six degrees of separation and then pinpointing the most important nodes for each degree based on connectivity and average distance. The results are presented in a comparative format, showing not only the most important nodes for each degree but also their reachability and average distance in other degrees. This approach allows for a thorough understanding of each node's significance across the network, testing the hypothesis about their consistent or variable importance in the graph's structure.

Output

```
Most important node for degree 1: Node 12220
Number of reachable nodes in degree 1: 4
Average distance in degree 1: 0.89

Number of reachable nodes and average distances for Node 12220 for the other degrees:
  - Degree 2: 5 reachable nodes, Average Distance: 5.09
  - Degree 3: 8 reachable nodes, Average Distance: 20.22
  - Degree 4: 13 reachable nodes, Average Distance: 54.48
  - Degree 5: 21 reachable nodes, Average Distance: 73.76
  - Degree 6: 31 reachable nodes, Average Distance: 105.89
--------------------------------------------------------------------------------
Most important node for degree 2: Node 2477
Number of reachable nodes in degree 2: 7
Average distance in degree 2: 7.84

Number of reachable nodes and average distances for Node 2477 for the other degrees:
  - Degree 1: 4 reachable nodes, Average Distance: 2.59
  - Degree 3: 10 reachable nodes, Average Distance: 21.99
  - Degree 4: 8 reachable nodes, Average Distance: 37.61
  - Degree 5: 14 reachable nodes, Average Distance: 82.10
  - Degree 6: 15 reachable nodes, Average Distance: 96.67
--------------------------------------------------------------------------------
Most important node for degree 3: Node 3267
Number of reachable nodes in degree 3: 11
Average distance in degree 3: 23.68

Number of reachable nodes and average distances for Node 3267 for the other degrees:
  - Degree 1: 2 reachable nodes, Average Distance: 6.22
  - Degree 2: 5 reachable nodes, Average Distance: 11.99
  - Degree 4: 13 reachable nodes, Average Distance: 47.54
  - Degree 5: 16 reachable nodes, Average Distance: 79.62
  - Degree 6: 24 reachable nodes, Average Distance: 106.31
--------------------------------------------------------------------------------
Most important node for degree 4: Node 15639
Number of reachable nodes in degree 4: 18
Average distance in degree 4: 41.38

Number of reachable nodes and average distances for Node 15639 for the other degrees:
  - Degree 1: 2 reachable nodes, Average Distance: 2.39
  - Degree 2: 4 reachable nodes, Average Distance: 11.59
  - Degree 3: 9 reachable nodes, Average Distance: 25.04
  - Degree 5: 20 reachable nodes, Average Distance: 58.72
  - Degree 6: 23 reachable nodes, Average Distance: 86.68
--------------------------------------------------------------------------------
Most important node for degree 5: Node 8447
Number of reachable nodes in degree 5: 19
Average distance in degree 5: 51.27

Number of reachable nodes and average distances for Node 8447 for the other degrees:
  - Degree 1: 3 reachable nodes, Average Distance: 8.08
  - Degree 2: 5 reachable nodes, Average Distance: 20.34
  - Degree 3: 6 reachable nodes, Average Distance: 29.10
  - Degree 4: 12 reachable nodes, Average Distance: 40.90
  - Degree 6: 14 reachable nodes, Average Distance: 62.75
--------------------------------------------------------------------------------
Most important node for degree 6: Node 627
Number of reachable nodes in degree 6: 25
Average distance in degree 6: 72.25

Number of reachable nodes and average distances for Node 627 for the other degrees:
  - Degree 1: 2 reachable nodes, Average Distance: 6.68
  - Degree 2: 6 reachable nodes, Average Distance: 19.73
  - Degree 3: 11 reachable nodes, Average Distance: 32.96
  - Degree 4: 15 reachable nodes, Average Distance: 47.76
  - Degree 5: 17 reachable nodes, Average Distance: 62.46
--------------------------------------------------------------------------------
```

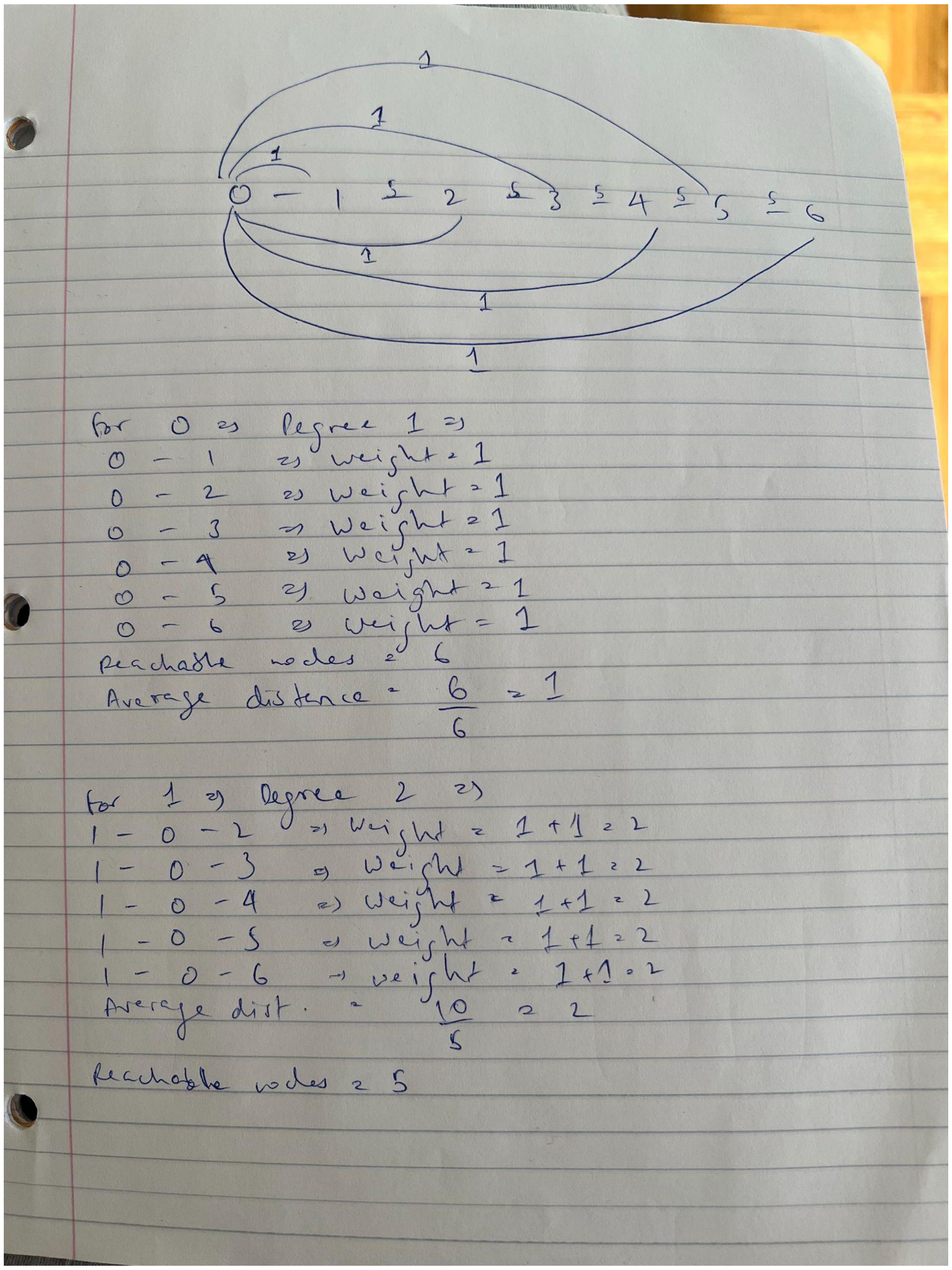
Conclusion from output

This output refutes my hypothesis that an important node in 1 degree is also an important node in the next degree. This is because of city planning around existing landscapes, nature or even man-made areas that need to be worked around.

Tests

- Function: analyze_by_degrees_of_separation

My tests were quite tricky to come up with because I not only needed to ensure that it was calculating the shortest distance to each node, it also had to ensure that it was within the right number of degrees. So, I came up with the following graph:

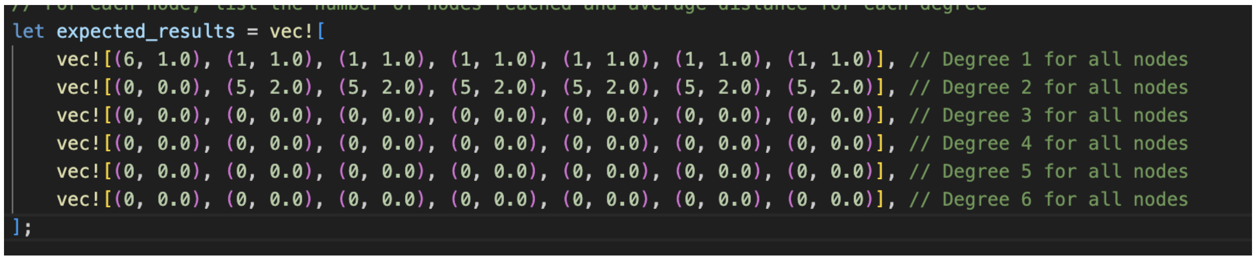

1
1
1
0 — 1 5 2 5 3 5 4 5 5 5 6
1
1
1

for 0 => Degree 1 =>
0 - 1 => weight = 1
0 - 2 => weight = 1
0 - 3 => weight = 1
0 - 4 => weight = 1
0 - 5 => weight = 1
0 - 6 => weight = 1
Reachable nodes = 6
Average distance = $\frac{6}{6} = 1$

for 1 => Degree 2 =>
1 - 0 - 2 => weight = 1 + 1 = 2
1 - 0 - 3 => weight = 1 + 1 = 2
1 - 0 - 4 => weight = 1 + 1 = 2
1 - 0 - 5 => weight = 1 + 1 = 2
1 - 0 - 6 => weight = 1 + 1 = 2
Average dist. = $\frac{10}{5} = 2$

Reachable nodes = 5

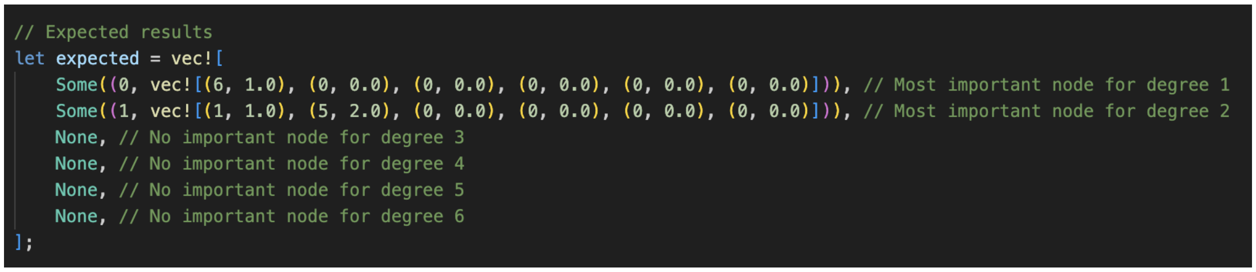
This graph ensured that for degree 1, the node 0 must be the most important because it is the most reachable and has the shortest average distances. To test dijkstra's algorithm the average distance of degree 0 should be 1, the algorithm should not reach the nodes 2 to 6 through the chain.



```
// For each node, list the number of nodes reached and average distance for each degree
let expected_results = vec![
    vec![(6, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0)], // Degree 1 for all nodes
    vec![(0, 0.0), (5, 2.0), (5, 2.0), (5, 2.0), (5, 2.0), (5, 2.0), (5, 2.0)], // Degree 2 for all nodes
    vec![(0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)], // Degree 3 for all nodes
    vec![(0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)], // Degree 4 for all nodes
    vec![(0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)], // Degree 5 for all nodes
    vec![(0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)], // Degree 6 for all nodes
];
```

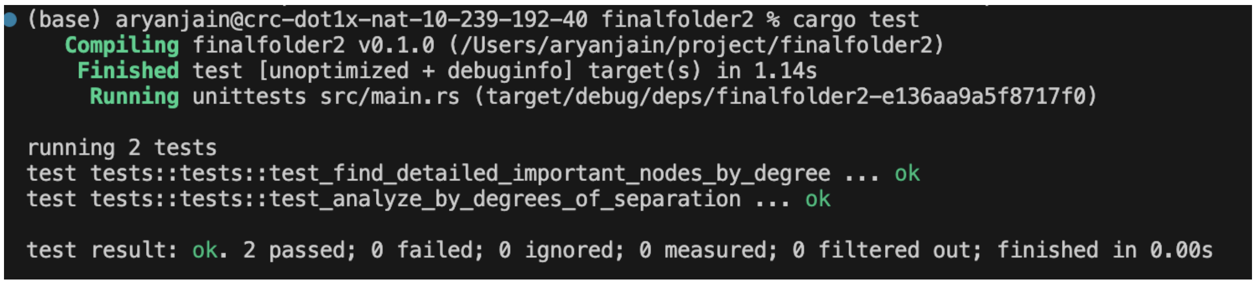
This is how the results should be. There are no results for degrees 3 to 6 because there are no nodes that reach this degree.

- Function: analyze_by_degrees_of_separation



```
// Expected results
let expected = vec![
    Some((0, vec![(6, 1.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)])), // Most important node for degree 1
    Some((1, vec![(1, 1.0), (5, 2.0), (0, 0.0), (0, 0.0), (0, 0.0), (0, 0.0)])), // Most important node for degree 2
    None, // No important node for degree 3
    None, // No important node for degree 4
    None, // No important node for degree 5
    None, // No important node for degree 6
];
```

For this function, it takes the top 10% of the most reachable nodes and must take the one with the shortest average distance. For degree 1 it should be node 0 because of the shortest average distance, and for degree 2 it should be 1. For the other degrees the function must return None.



```
(base) aryanjain@crc-dot1x-nat-10-239-192-40 finalfolder2 % cargo test
   Compiling finalfolder2 v0.1.0 (/Users/aryanjain/project/finalfolder2)
    Finished test [unoptimized + debuginfo] target(s) in 1.14s
     Running unittests src/main.rs (target/debug/deps/finalfolder2-e136aa9a5f8717f0)

running 2 tests
test tests::tests::test_find_detailed_important_nodes_by_degree ... ok
test tests::tests::test_analyze_by_degrees_of_separation ... ok

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

Here we can see that the tests have passed.

Conclusion

In conclusion, the project runs smoothly with around 2 minutes of time required to execute, it provides a clean output that helps interpret the most important nodes for each degree of separation and how important they are in other degrees. I do realize that I have taken an arbitrary percentage of 10% to calculate the top nodes. I have done this to ensure I give importance to the reachable nodes and the average distance. I look to pursue the project further to find a more accurate percentage and analyze deeper into the different road networks in cities, counties, states and countries.