DS210 Write up

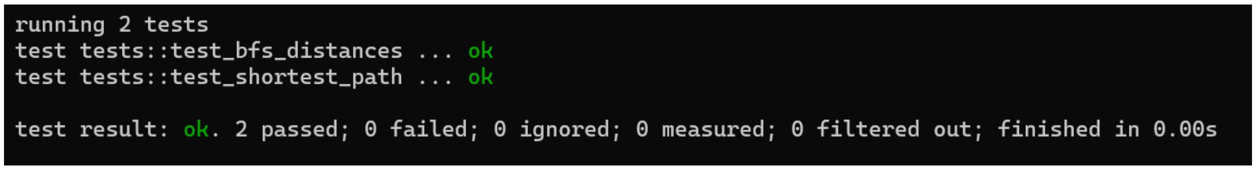
This code implements functions for analyzing a co-occurrence network of recipe ingredients. I used main functions like HashMap, VecDeque, file and BufRead. HashMaps in my code makes storage and retrieval of ingredient relationships, enabling graph operations such as adding edges, BFS traversal, and shortest path calculation in the context of analyzing the co-occurrence network of recipe ingredients. In addition, I used VecDeque because it is essential to perform BFS traversal and I used File to open and read data from a file named "ingredients.txt". Lastly, I used BufRead to efficiently read the data from the file

Firstly, I performed a graph structure, where vertices represent ingredients and edges represent co-occurrences between ingredients in recipes. The graph is implemented using an adjacency list representation, where each vertex (ingredient) is associated with a set of neighboring vertices (ingredients it co-occurs with). I also added edges which allows a directed edge between two ingredients in the graph, showing a co-occurrence relationship between them in at least one recipe. I performed a BFS traversal which helps explore how ingredients are connected in recipes. It starts from one ingredient and checks which other ingredients are directly or indirectly linked to it through shared recipes. For example, one of the outputs from this was "2987: 2" which is in the format of "ingredient id: frequency. Overall, this method analyzes ingredient usage. From BFS, I calculated the average distance from a specified starting ingredient vertex to all other reachable ingredient vertices in the graph. The output I got for this was “Average distance from node 0: 1.7999534233814625.” So my starting point was node 0 and 1.7999534233814625 is the average distance from node 0 to all other reachable nodes in the graph. Essentially, this provides a measure of how ingredients are interconnected within the co-occurrence network. Furthermore, I also used BFS to find the shortest path between two specified ingredient vertices, which were 0 and 400. The output "Shortest path from 0 to 400: [0, 207, 400]" means the quickest way to go from ingredient 0 to ingredient 400 in the recipe network. You'd go from ingredient 0 to ingredient 207, and then to ingredient 400. If there was no connection, it means no path exists between the specified ingredients.

In the main function, the code reads the graph data from the file named "ingredients.txt" and constructs a graph. It then performs BFS traversal, calculates the average distance, and finds the shortest path between specified ingredients, which show results on console after using cargo run.

The provided test cases ensure the correctness of BFS traversal and shortest path finding functionalities for the graph structure, validating that the implemented methods work as expected in analyzing the co-occurrence network of recipe ingredients.

Result of tests:



```
running 2 tests
test tests::test_bfs_distances ... ok
test tests::test_shortest_path ... ok

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