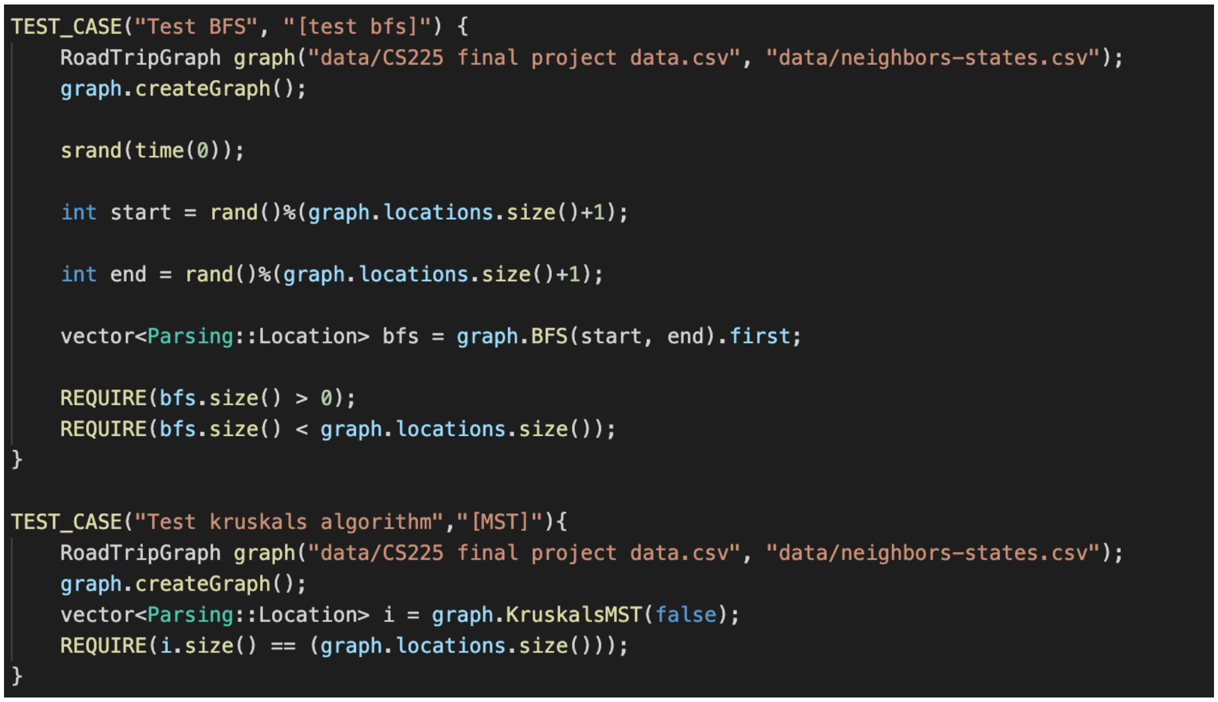
CS225 Final Project- Results

Final Deliverables:

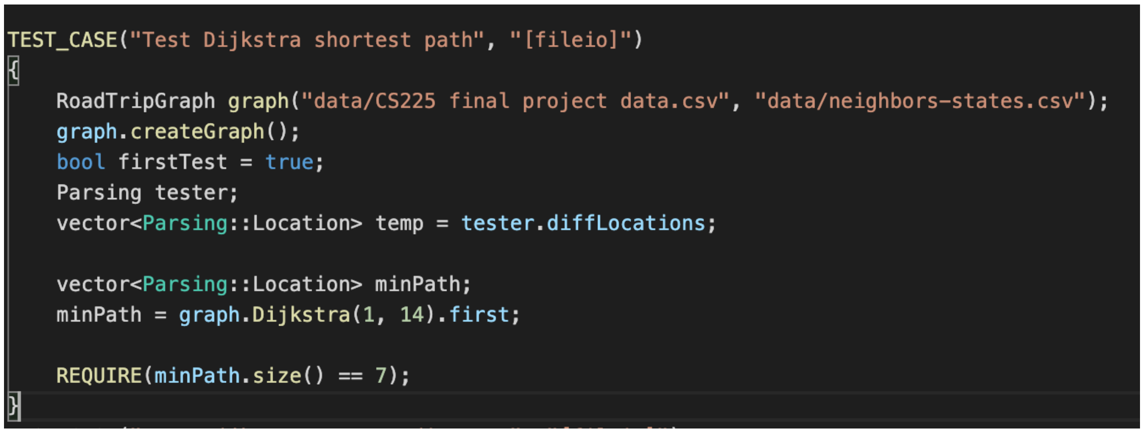
In our proposal we sought out to be able to create a road-trip planner which could successfully find the quickest route from point A to point B using metrics for distance and cost. In our final implementation, we were able to do this, but were not able to implement the cost as well. Overall, in order to make our project successful, we implemented Kruskal's Algorithm, DIjktra's Algorithm, and Breadth First Search. Through writing each function, we made various observations, and the results were very similar to what we expected. In this project, the final deliverable is the program run in main. It takes a starting and ending state, and delivers the path given by BFS and Dijkstra's, while also printing the Minimum Spanning Tree made by Kruskal's. We also printed the shortest path onto a PNG as a part of the output.

For each algorithm and the parsing function to put the data into C++, we wrote tests which checked whether the algorithms returned reasonable results, and whether the input data was parsed correctly in the first place.

Although we found it a little difficult to write really specific tests because the graph was very dense, the tests we wrote did pass, and there will be pictures below.



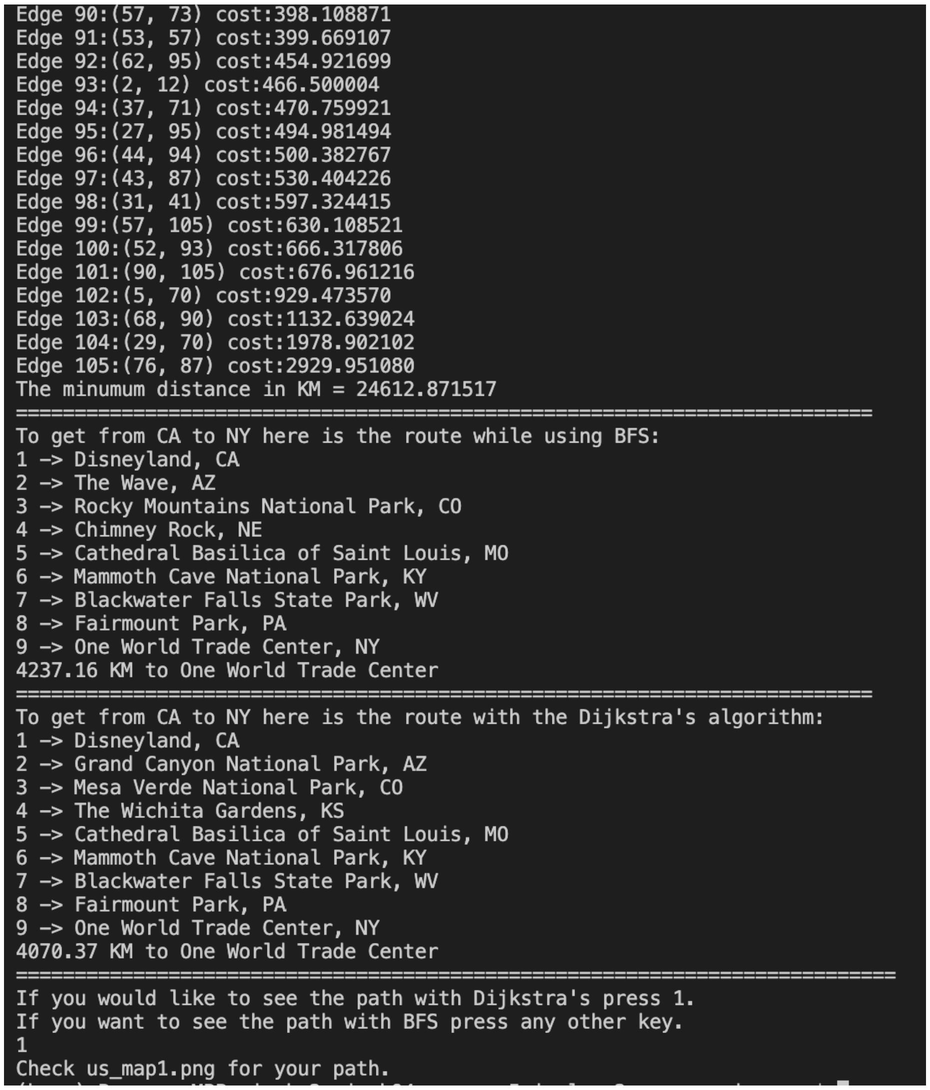
```
TEST_CASE("Test BFS", "[test bfs]") {
    RoadTripGraph graph("data/CS225 final project data.csv", "data/neighbors-states.csv");
    graph.createGraph();

    srand(time(0));

    int start = rand()%(graph.locations.size()+1);

    int end = rand()%(graph.locations.size()+1);

    vector<Parsing::Location> bfs = graph.BFS(start, end).first;

    REQUIRE(bfs.size() > 0);
    REQUIRE(bfs.size() < graph.locations.size());
}

TEST_CASE("Test kruskals algorithm","[MST]"){
    RoadTripGraph graph("data/CS225 final project data.csv", "data/neighbors-states.csv");
    graph.createGraph();
    vector<Parsing::Location> i = graph.KruskalsMST(false);
    REQUIRE(i.size() == (graph.locations.size()));
}
```



```
TEST_CASE("Test Dijkstra shortest path", "[fileio]")
{
    RoadTripGraph graph("data/CS225 final project data.csv", "data/neighbors-states.csv");
    graph.createGraph();
    bool firstTest = true;
    Parsing tester;
    vector<Parsing::Location> temp = tester.diffLocations;

    vector<Parsing::Location> minPath;
    minPath = graph.Dijkstra(1, 14).first;

    REQUIRE(minPath.size() == 7);
}
```

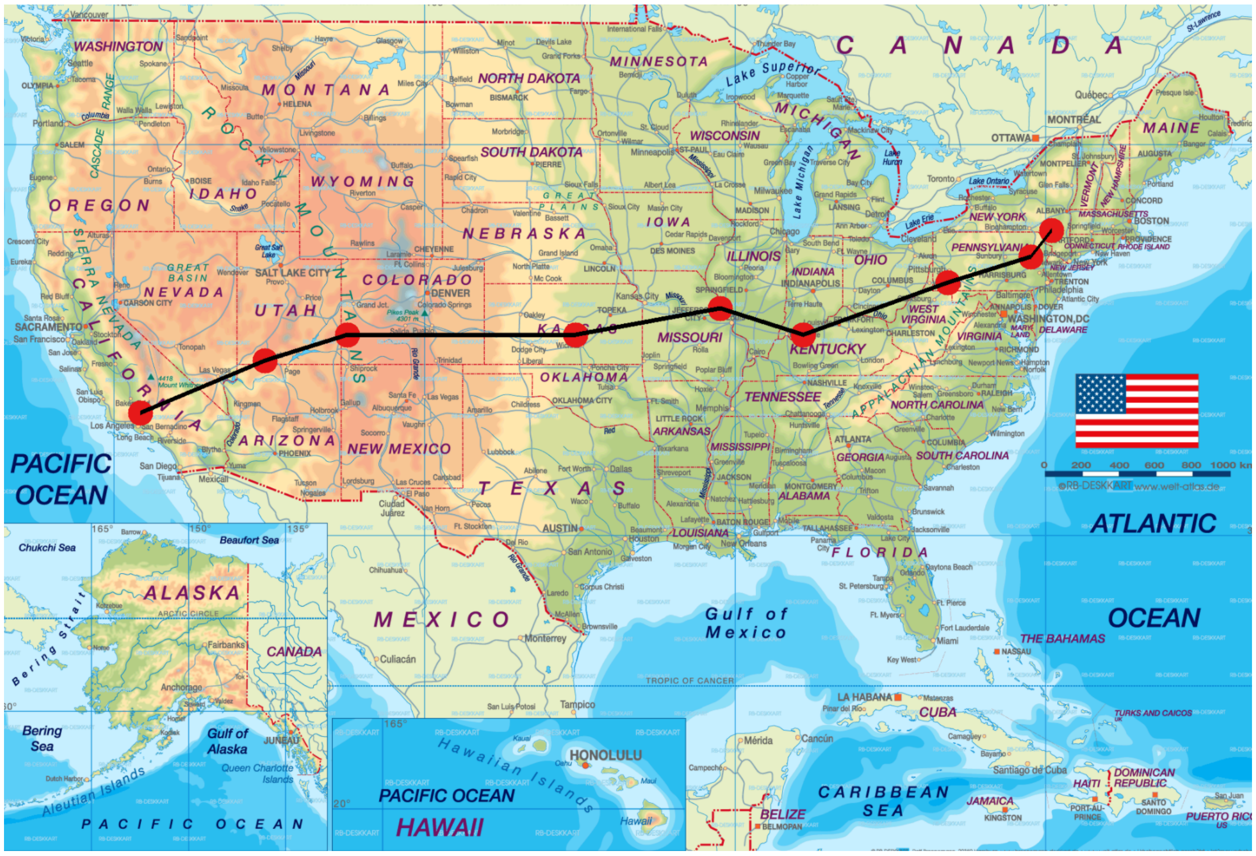
Besides writing test suites, the easy way to see the final result is to just run the main function.

For the input to our algorithms we took in a state, and randomly picked one attraction in the state to start at and did the same for the end point. When ran from CA to NY the following output is returned:



```
Edge 90:(57, 73) cost:398.108871
Edge 91:(53, 57) cost:399.669107
Edge 92:(62, 95) cost:454.921699
Edge 93:(2, 12) cost:466.500004
Edge 94:(37, 71) cost:470.759921
Edge 95:(27, 95) cost:494.981494
Edge 96:(44, 94) cost:500.382767
Edge 97:(43, 87) cost:530.404226
Edge 98:(31, 41) cost:597.324415
Edge 99:(57, 105) cost:630.108521
Edge 100:(52, 93) cost:666.317806
Edge 101:(90, 105) cost:676.961216
Edge 102:(5, 70) cost:929.473570
Edge 103:(68, 90) cost:1132.639024
Edge 104:(29, 70) cost:1978.902102
Edge 105:(76, 87) cost:2929.951080
The minumum distance in KM = 24612.871517
=========================================================================
To get from CA to NY here is the route while using BFS:
1 -> Disneyland, CA
2 -> The Wave, AZ
3 -> Rocky Mountains National Park, CO
4 -> Chimney Rock, NE
5 -> Cathedral Basilica of Saint Louis, MO
6 -> Mammoth Cave National Park, KY
7 -> Blackwater Falls State Park, WV
8 -> Fairmount Park, PA
9 -> One World Trade Center, NY
4237.16 KM to One World Trade Center
=========================================================================
To get from CA to NY here is the route with the Dijkstra's algorithm:
1 -> Disneyland, CA
2 -> Grand Canyon National Park, AZ
3 -> Mesa Verde National Park, CO
4 -> The Wichita Gardens, KS
5 -> Cathedral Basilica of Saint Louis, MO
6 -> Mammoth Cave National Park, KY
7 -> Blackwater Falls State Park, WV
8 -> Fairmount Park, PA
9 -> One World Trade Center, NY
4070.37 KM to One World Trade Center
===========================================================================
If you would like to see the path with Dijkstra's press 1.
If you want to see the path with BFS press any other key.
1
Check us_map1.png for your path.
```

The map output looks like this:



Discoveries:

We ran into a few problems and made a few discoveries throughout the course of this project. But I think they can be condensed to just two things. First of all, when writing Dijkstra's we ended up getting paths which were not efficient and sometimes even longer than BFS. However, we realized that we weren't updating the weights of the vertices in the queue everytime they were added, which led to less efficient minimum routes. We also discovered that we needed to have an attraction in each state in order to find the most efficient path in the graph. When that condition was not met, we were met with errors, bad routes, and more. Overall, there were a few major issues we faced, but persevered to get a working result.

Final Thoughts:

We thought that the results were very interesting, and were surprised at the efficiency and ingenuity and some of the algorithms we were implementing. Although we faced some problems during the project, the results were pretty compelling. A nice addition would have been to add more factors like cost, time to travel, and more. Overall we accomplished everything we sought out to, but feel like we could have done even more interesting things with the data.