# Approach, Design & Algorithm

Before starting this project, as always, I took a good look at the project specifications and any relevant documentation to try and grasp exactly what the constraints and objectives were. Overall, the project was fairly similar to the graph implementation discussed in the book the only major difference was the requirement that each road object has to have two endpoints. I am not entirely sure why this choice was made, other than to make the project different, but I began planning how I could work around this necessity. After altering my plan a little bit I began working on the lowest-level classes and then working my way up the ladder.

For the Town class, I began by implementing the methods specified in the Javadoc to the best of my ability. I believe there were a few methods I left until later because I understood their implementation could be gravely impacted by project specifications.

I followed a similar process for the Road class initially only specifying the methods mentioned in the java docs and project specifications. I believe there were a few methods I left out until I got deeper into the project as they were more complicated and depended on the other aspects of the project.

At the heart of the project, the graph class took the most time. Many methods in this class required deep consideration and brainstorming to determine a possible solution. I began by slowly working through all the methods which add, retrieve, remove, or alter objects in the graph. Essentially, I completed everything but the shortest path algorithm methods. I stopped before the shortest path algorithm to ensure I could thoroughly test the “fundamentals” of my graph before moving forward with an algorithm that would depend on the validity of these fundamentals alongside arguably the most complicated algorithm in the entire project.

Hence, I moved into the testing phase to ensure my fundamentals were “secure” before going back to complete the shortest path algorithm. This testing phase involved console testing, private JUnit testing, and running instructor-provided tests. After I went through and ironed out any bumps in my implementation of the fundamentals I then moved on to the TownGraphManager class so I could gain access to the more rigorous instructor test provided.

Initially, I only completed the methods which would overlay or wrap around the methods I had already written in the Graph class. Those are the methods that add, retrieve, remove, or alter objects in the graph. But, once again in this class, I ignored the shortest path algorithm until I had all the fundamentals down. Once I completed these methods I went through the second round of testing to ensure all my fundamentals were correct.

Then, I went back and worked on the shortest path algorithm. The shortest path algorithm was not too much of a pain in the neck. The most confusing part was why my priority queue was sorted in the wrong order. However, once I got that figured out my shortest path algorithm ran pretty smoothly.

From there, it was simply more testing, finishing up JUnit tests and doing some console-based testing as well.

# Learning Experience

## What did you learn?

I think this project was a great opportunity to review the concepts we had learned about in the graph module. However, I wouldn’t say I learned anything new, I just reinforced my prior understanding.

## What issues did you encounter, if any?

My issues mainly centered around project specifications. For instance, for the Road class, the compareTo method and the equals method do not agree upon which two objects are equal. But also, the compareTo method compares names which is useless in our implementation. Moreover, forcing Road objects to have two endpoints is redundant at best and leads to massive headaches at worst. As for many of the projects here at MC, a significant amount of time spent on these projects is devoted to working around poor project specifications. Do not take this as useless criticism. I know for certain that throughout my time at MC many of my peers have shared these concerns and I believe it would be best for the department to iterate on this feedback and use it to improve the projects we are given. I am sure there have been hundreds of students before me with the same worries and I am sure there will be hundreds more in the future if no changes are made.

## What would you have done differently?

I do not think I would make any changes on my end. I would love to do this project again but work on some sort of traveling salesman solution. Also, I would love to have a GUI that actually shows the graph itself rather than just showing text-based names of different components.

## How can you apply this concept in the future?

As alluded to earlier the concepts enforced in this project could be used to solve other graph theory problems. But also, each of these problems can be solved within a different context. Take for example the problem of connectivity, for a flight application, this could decern whether two cities are connected by some “path” of flights. In a social media application where each vertex represents a person on the platform and each edge represents a friendship, two vertices that are connected could be said to be in the same “social web.” I assume there are many more such variations of these problems and many more problems we have not yet learned about.

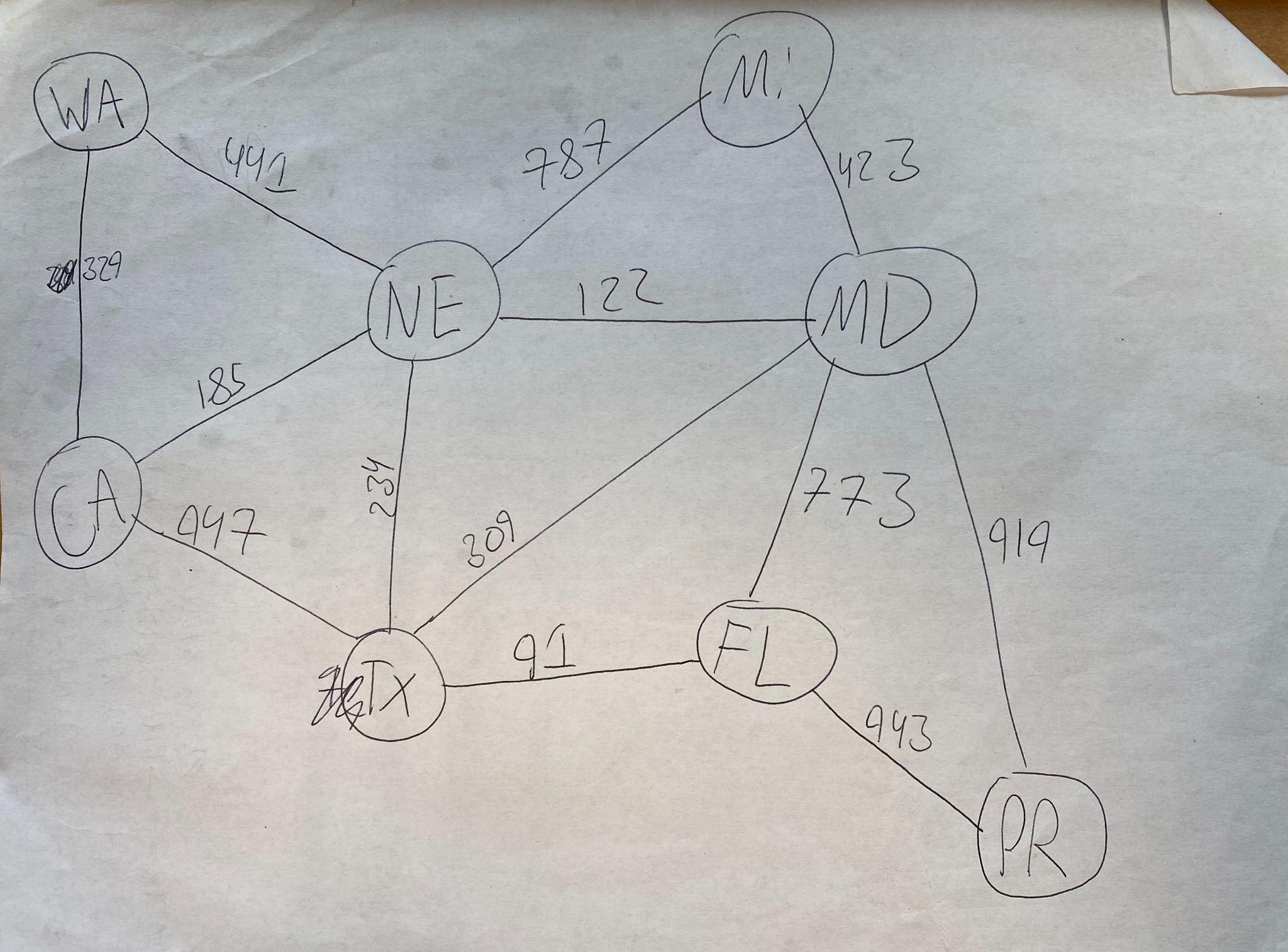
## Questions

I have no questions at this time.

## Assumptions

* Many of my assumptions are written within my code directly next to the code it impacts
* the graph is meant to be undirected
* there will be at max one edge between each pair of distinct vertices
  + that way each edge can be described distinctly by its endpoints
* town and road names should be case sensitive
* when reading a file the road name will contain no spaces
  + but the town names can have spaces

# Test Cases

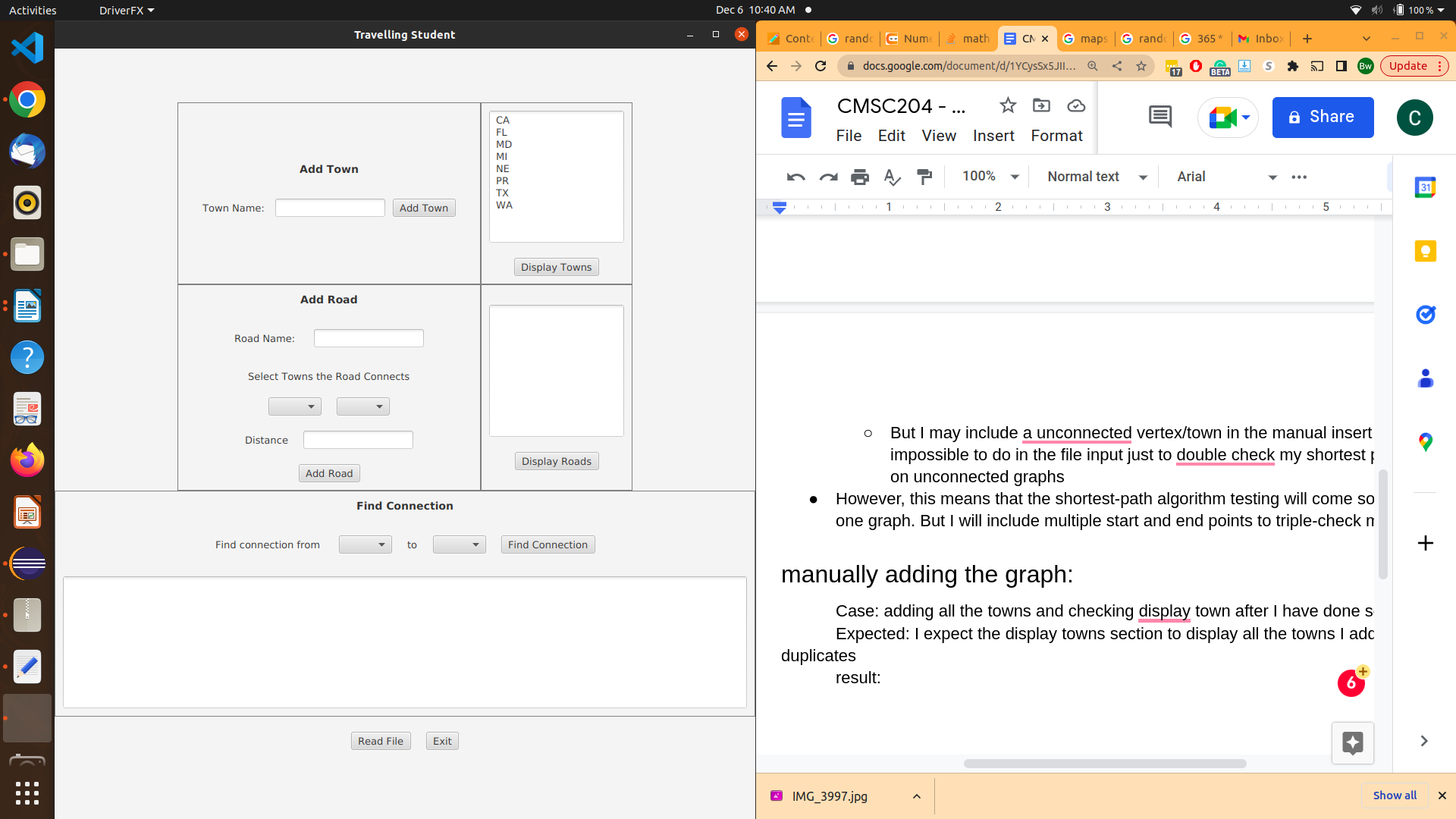
* For both of my test cases I will essentially be testing my program against this simply hand-made graph:
  + 
* I will test manually inserting this graph and I will create a text file for this graph to insert it into the program.
  + But I may include an unconnected vertex/town in the manual insert which is impossible to do in the file input just to double-check my shortest path algorithm on unconnected graphs
* However, this means that the shortest-path algorithm testing will come solely from this one graph. But I will include multiple start and end points to triple-check my algorithm
  + I will do half the testing in the manual input and half of the testing in the file input. I will try my best to have distinct test cases for both

## manually adding the graph:

Case: adding all the towns and checking display town after I have done so

Expected: I expect the display towns section to display all the towns I added with no duplicates

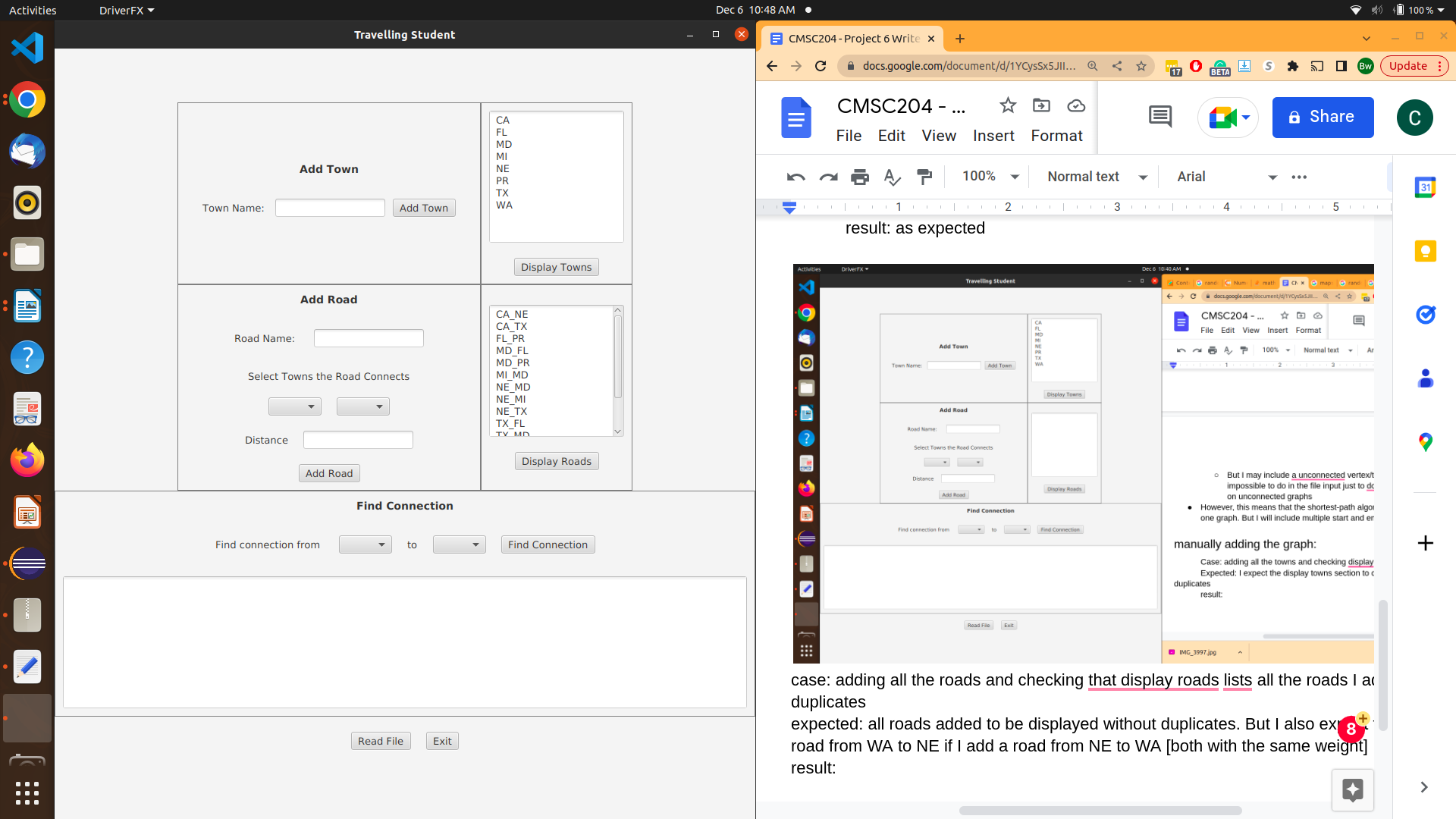
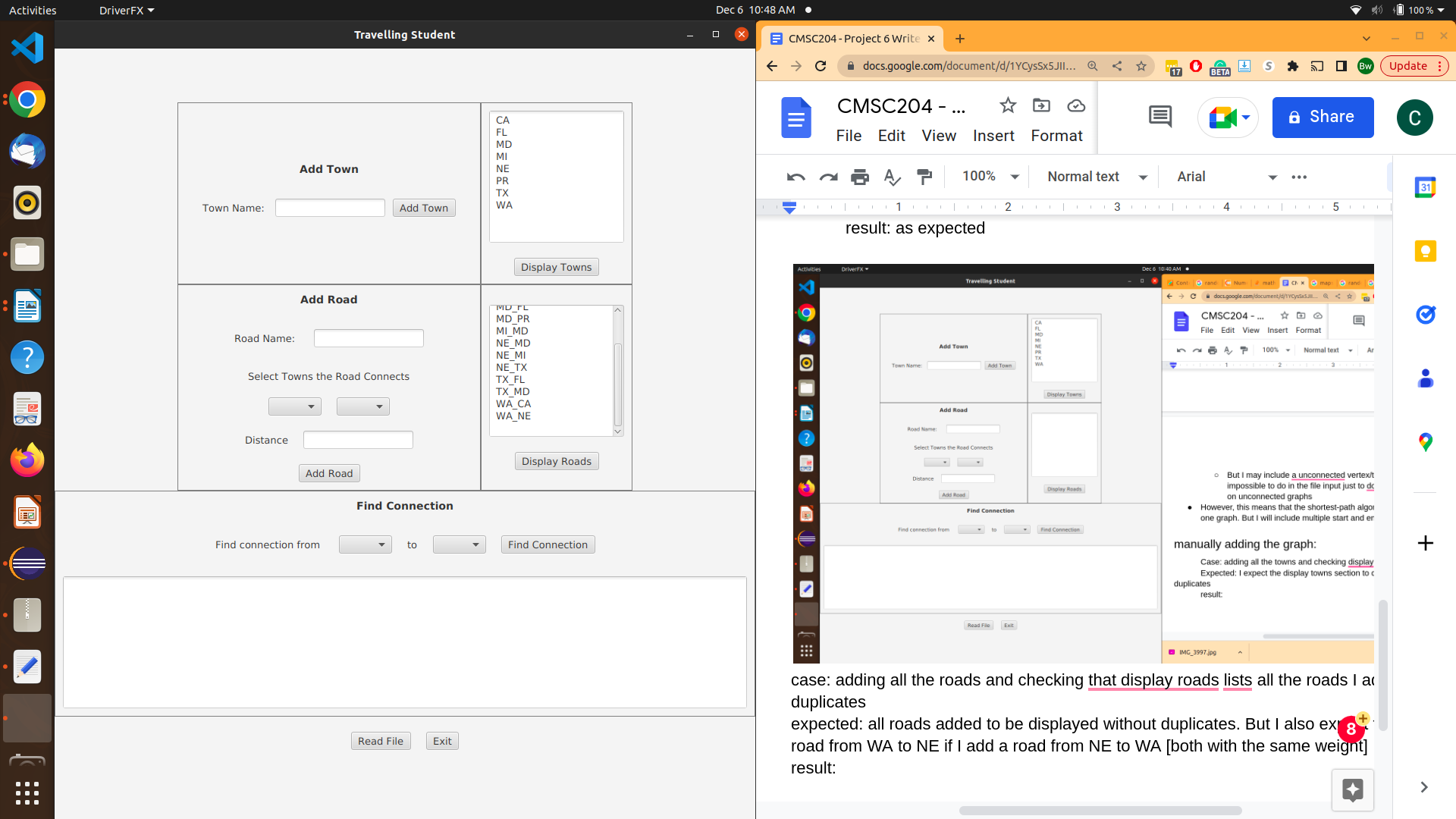
result: as expected



case: adding all the roads and checking that display roads lists all the roads I added without duplicates

expected: all roads added to be displayed without duplicates. But I also expect there to be a road from WA to NE if I add a road from NE to WA [both with the same weight]

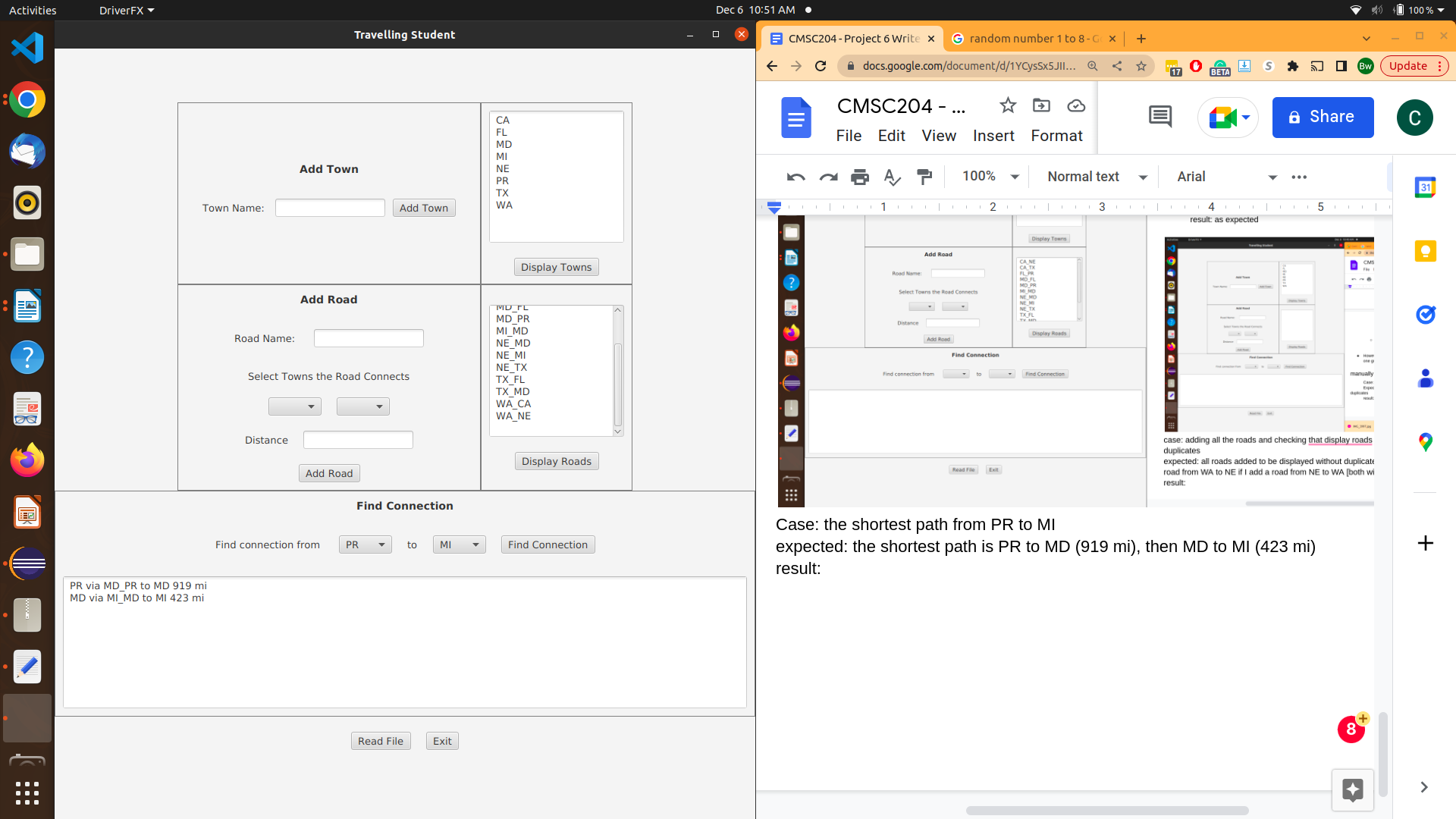
result: as expected all 13 roads were added



Case: the shortest path from PR to MI

expected: the shortest path is PR to MD (919 mi), then MD to MI (423 mi)

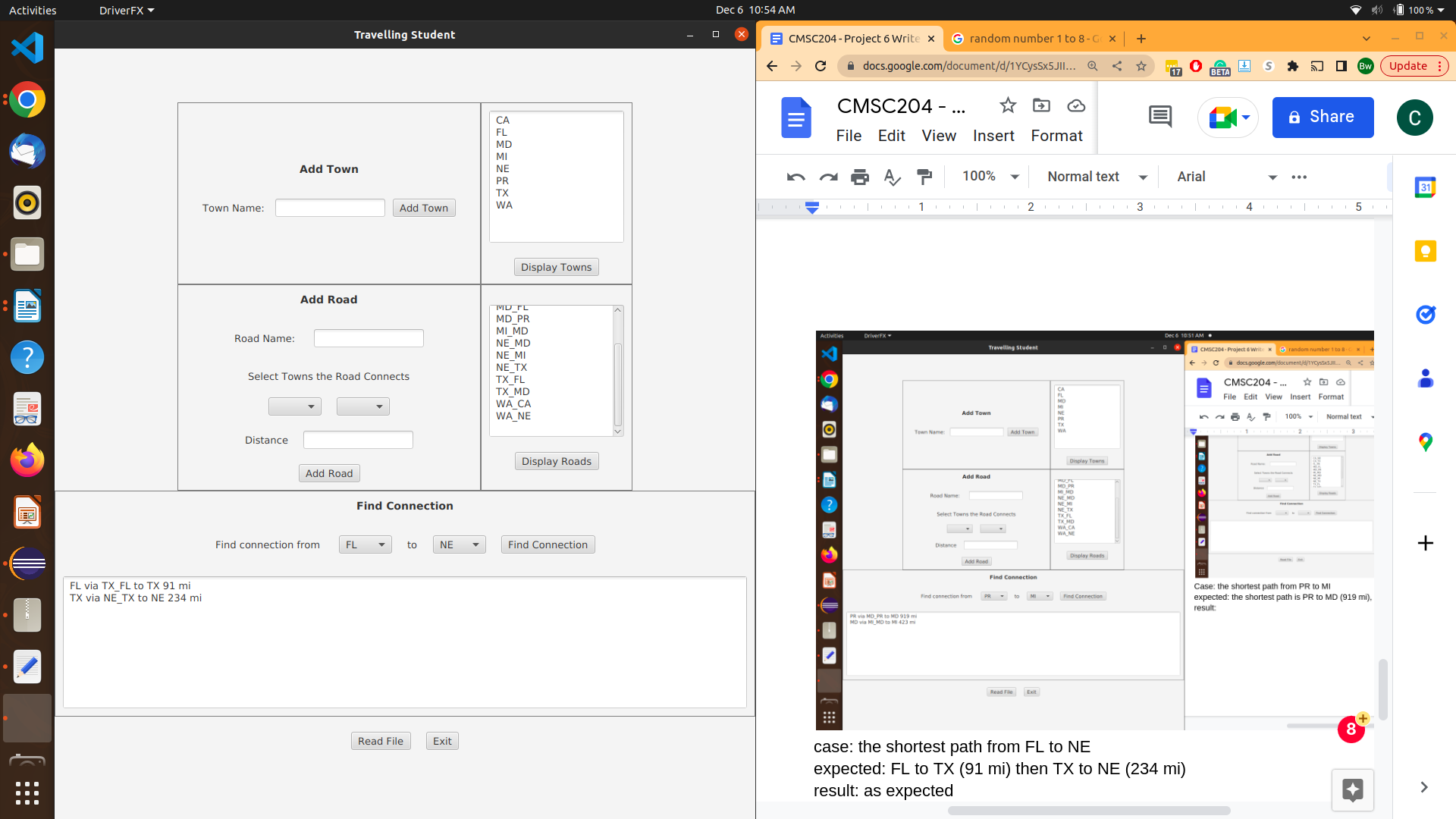
result: as expected



case: the shortest path from FL to NE

expected: FL to TX (91 mi) then TX to NE (234 mi)

result: as expected



case: the shortest path from MI to TX

expected: MI to MD (423 mi) then MD to TX (309 mi)

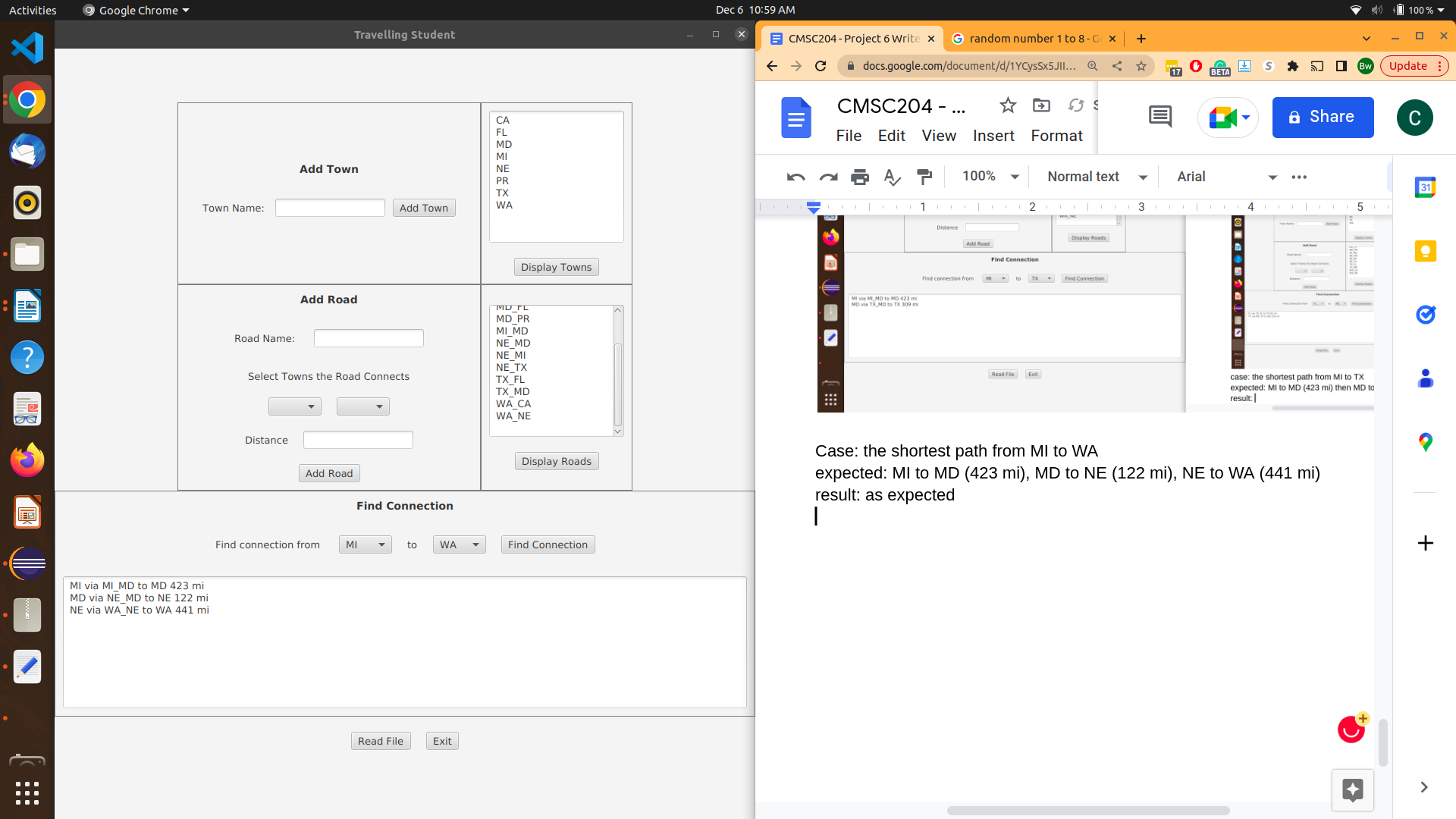
result: as expected



Case: the shortest path from MI to WA

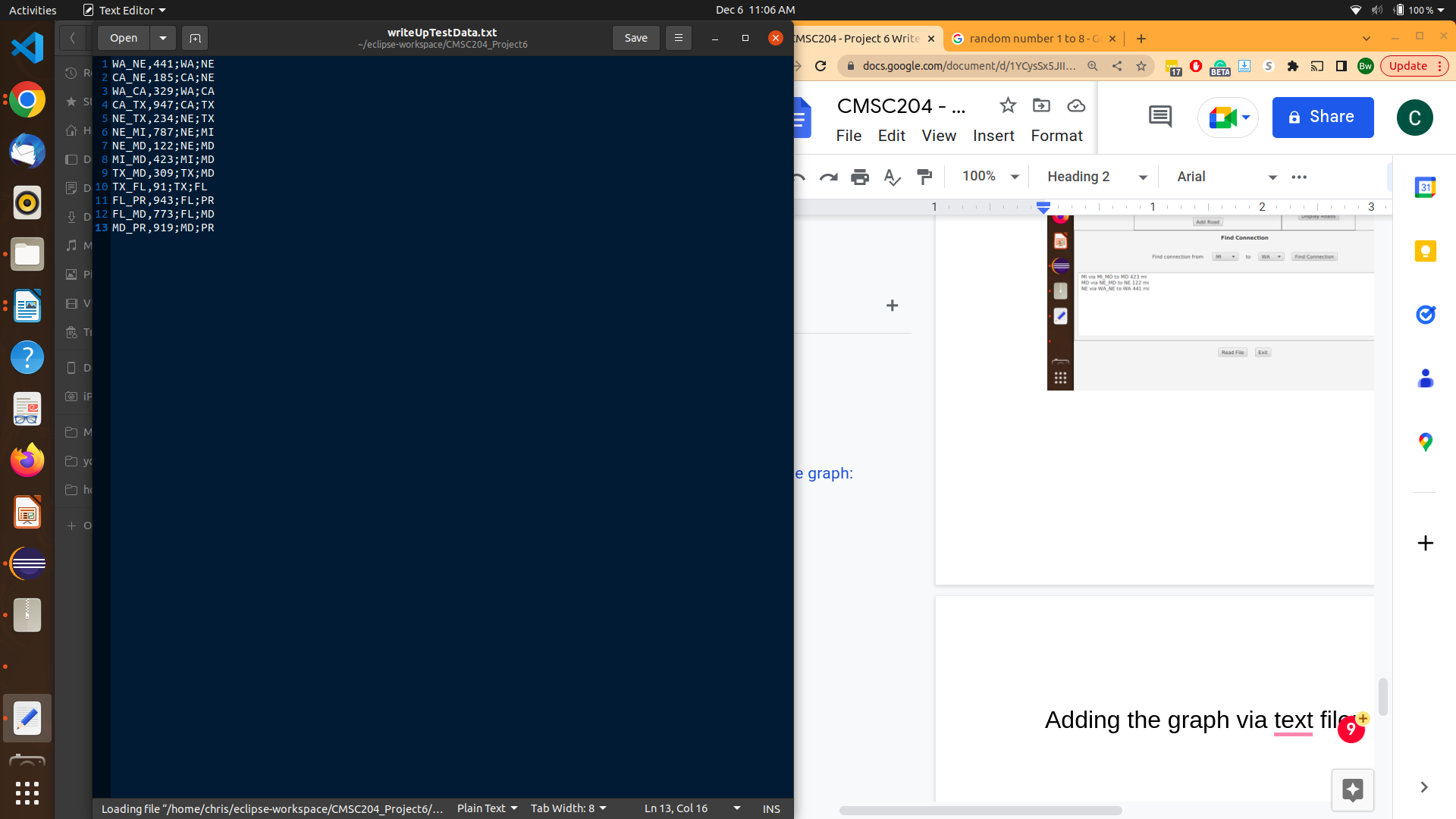
expected: MI to MD (423 mi), MD to NE (122 mi), NE to WA (441 mi)

result: as expected



## Adding the graph via a text file:

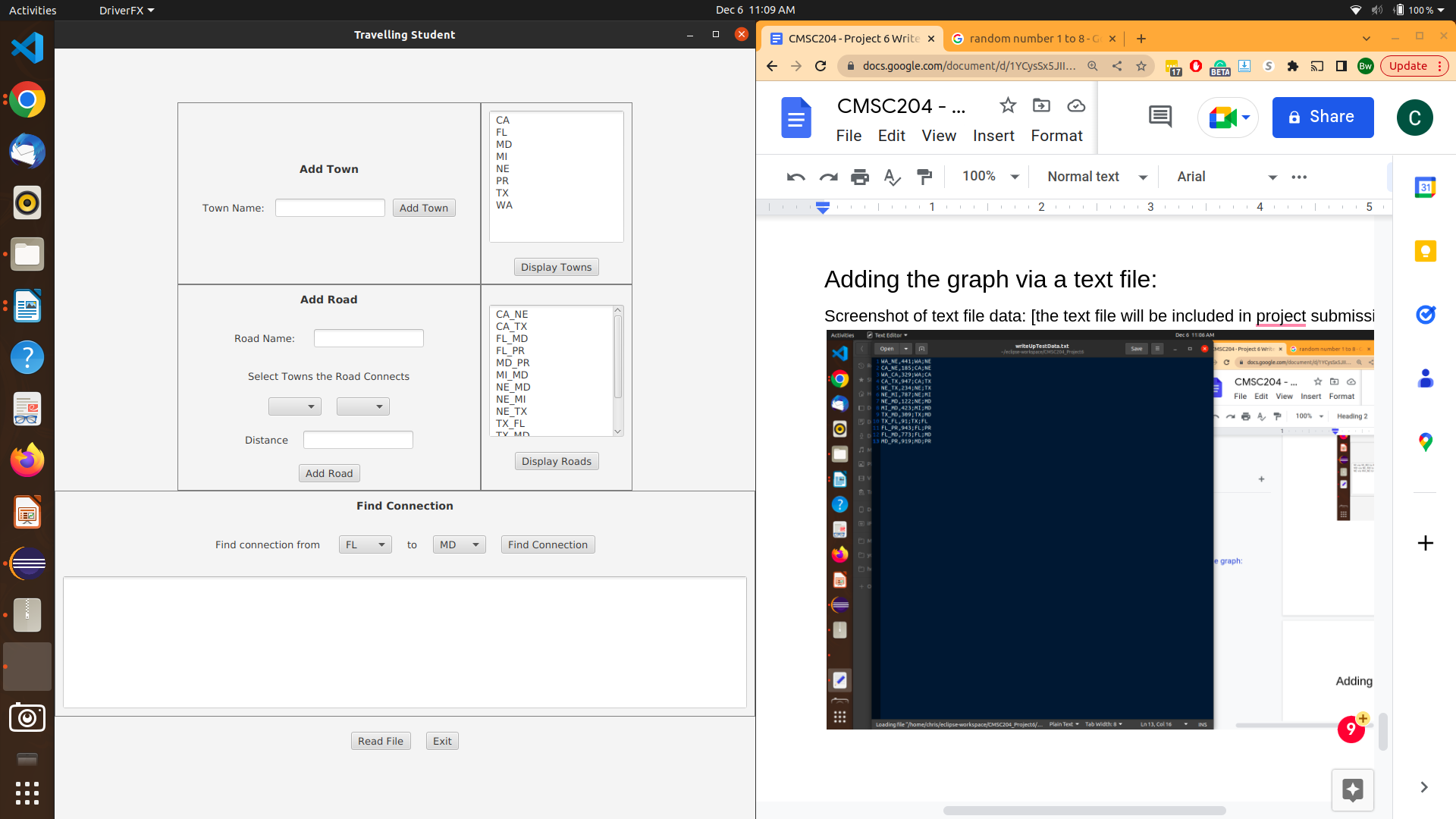
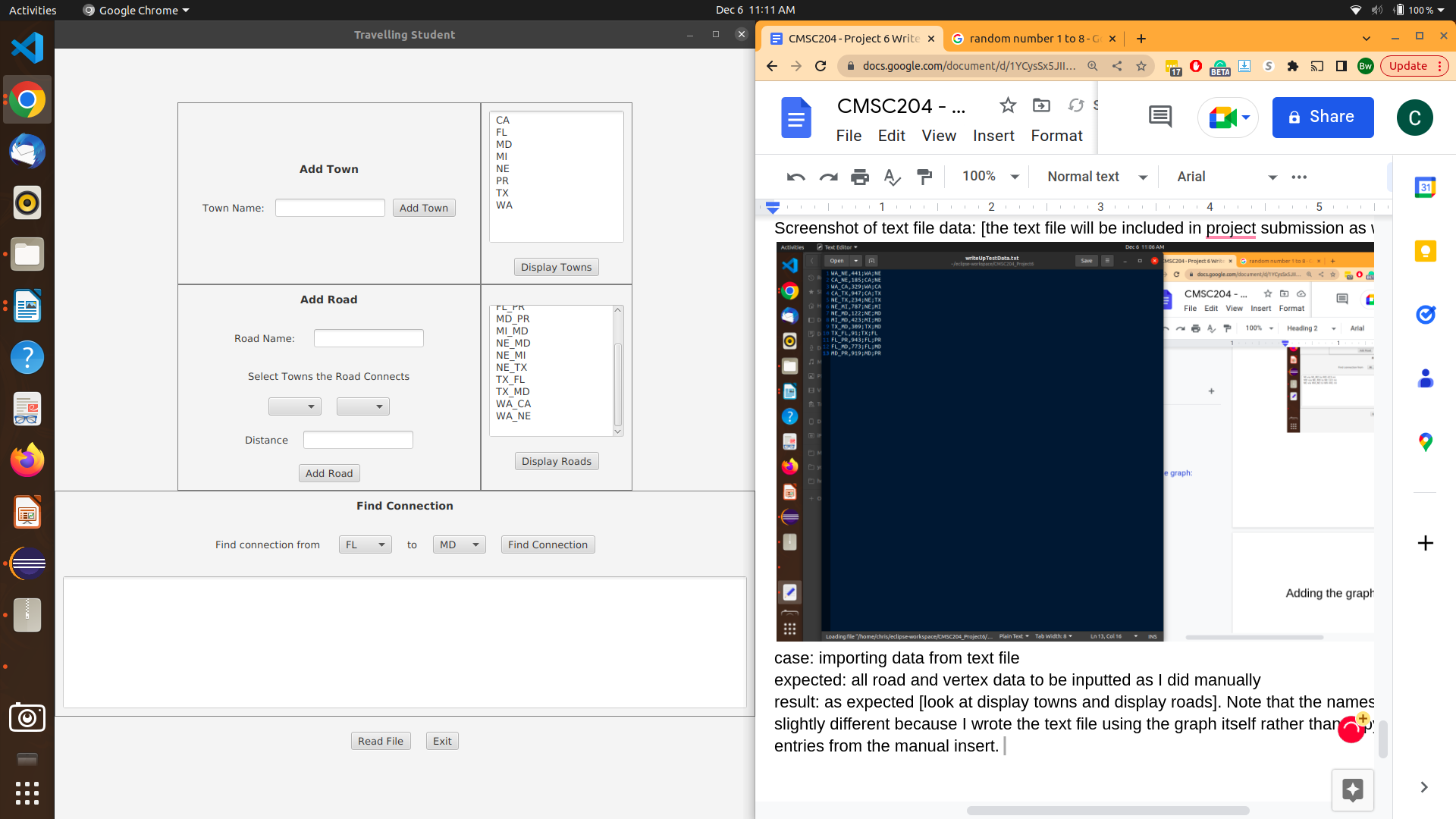
Screenshot of text file data: [the text file will be included in project submission as well]



case: importing data from text file

expected: all road and vertex data to be inputted as I did manually

result: as expected [look at display towns and display roads]. Note that the names might be slightly different because I wrote the text file using the graph itself rather than copying my entries from the manual insert.



the following few cases will test the shortest path algorithm further but using the data inported from the text file

case: the shortest path from MD to CA

expected: MD to NE (122 mi), then NE to CA (185 mi)

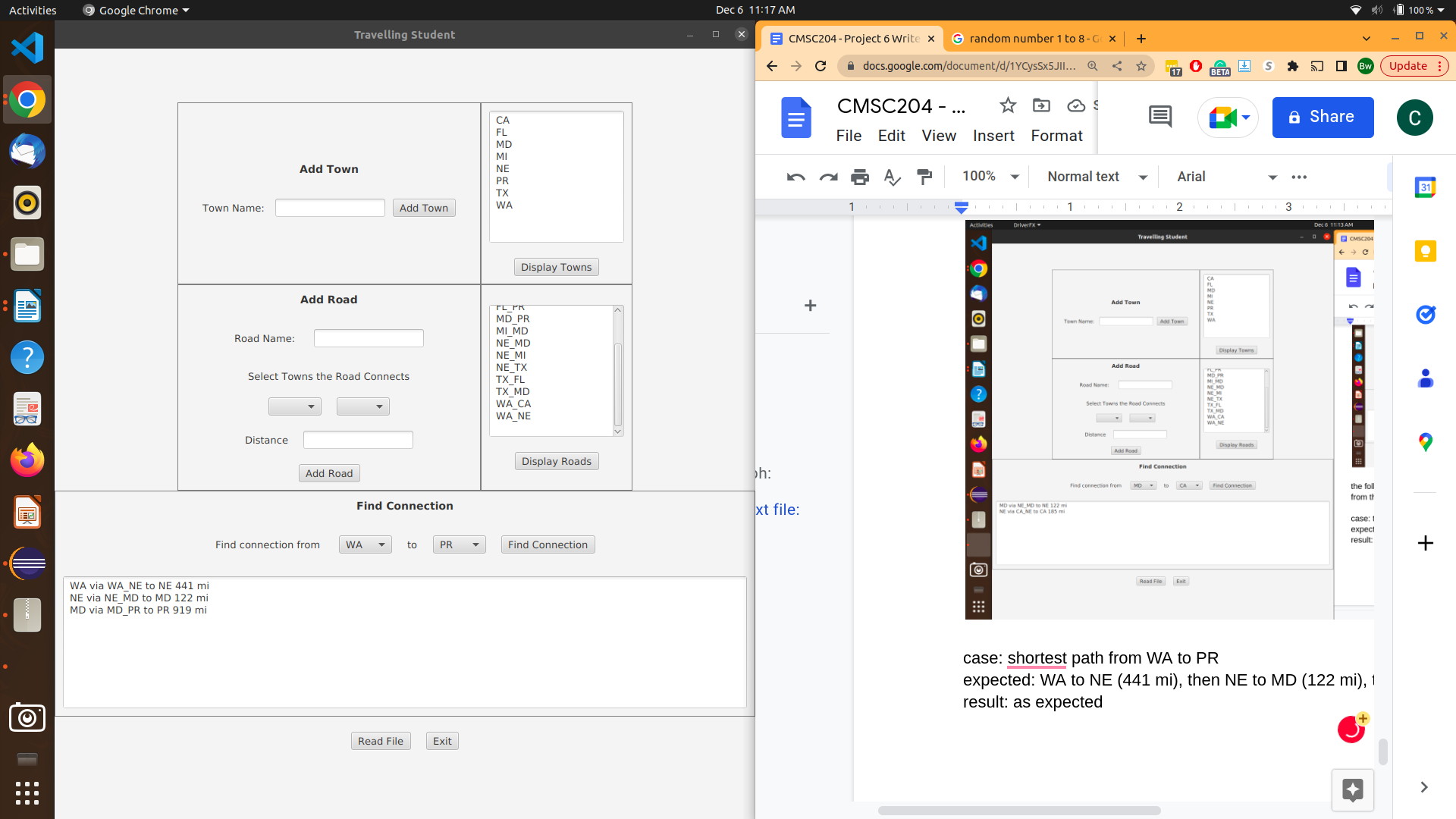
result: as expected



case: shortest path from WA to PR

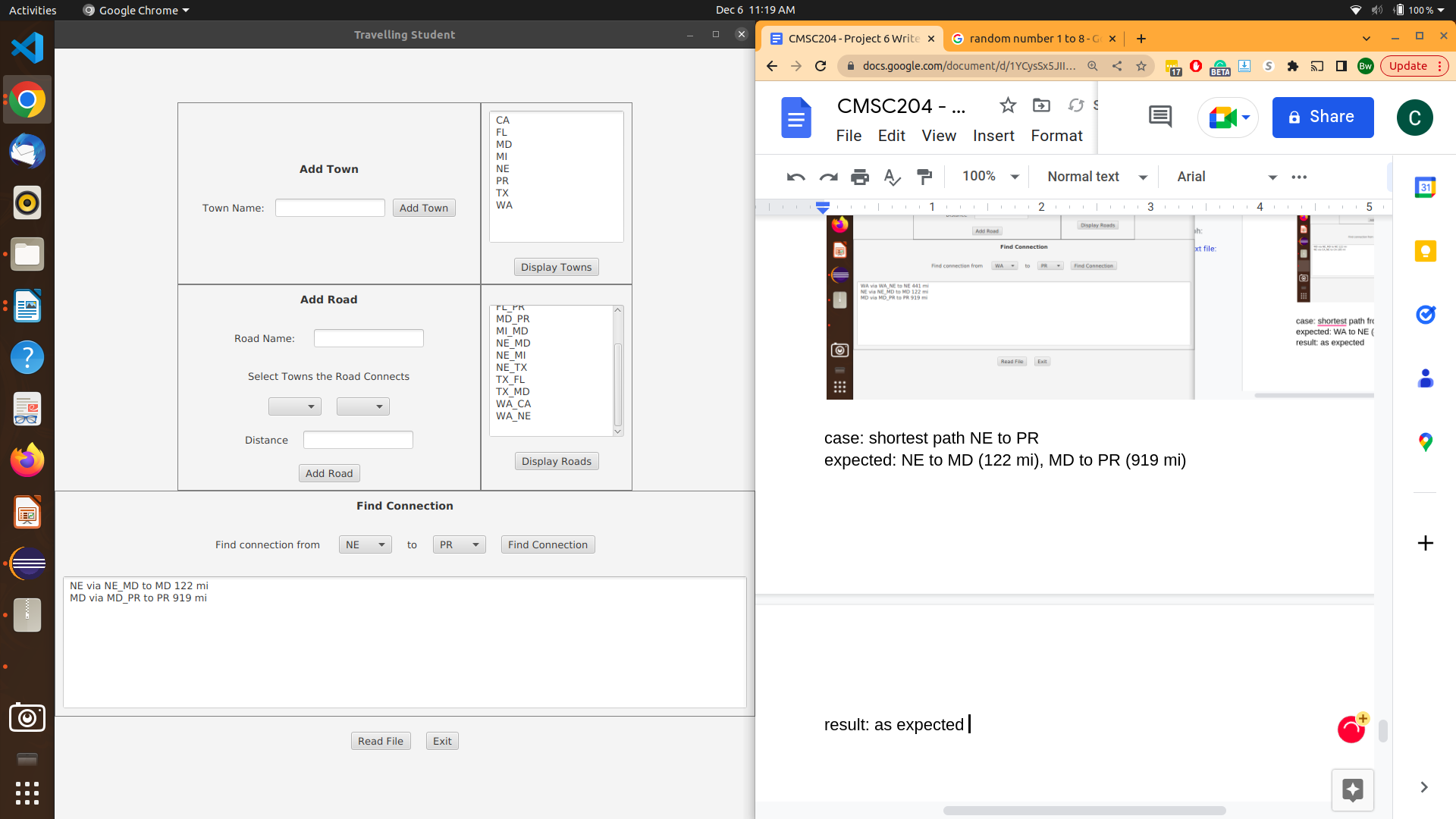
expected: WA to NE (441 mi), then NE to MD (122 mi), then MD to PR (919 mi)

result: as expected



case: shortest path NE to PR

expected: NE to MD (122 mi), MD to PR (919 mi)

result: as expected 

BELOW I MODIFY THE DATA A LITTLE BIT

case: add a town and see what happens when you try to find the shortest path to or from it

result: DE to PR spits out data for MI to PR I am going to investigate why this is occuring. I will first start by taking a look at the GUI and then move over to my code. So the GUI tests if the resultant ArrayList is empty but seems to return a nonempty arraylist when two points are not connected.

so I was able to fix this by conditionally adding the initial town into the stack and conditionally popping from the stack into the ArrayList dependent on if the stack was empty.

I will go through and check that all my JUnit tests work fine before proceeding. Yup this fixed the issue and did not affect other tests.

case: redo on the last case now that I fixed the issue. add a town that is unconnected and try to find the shortest path to or from it. [I will not include the screenshots for every town to the unconnected town and unconnected town to every other town. I will simply include one of each type but I took a look at all combinations on my end.]

expected: “you can’t get there from here”

result: as expected

