- Harm/Benefit Analysis: [Describe who is most likely to be harmed and who is most likely to benefit from this code, if it is used in the real world. Is the good greater than the harm? Would you be willing to implement it?]

Who is most likely to benefit are individuals who want to get to places quickly and who are able to take smaller roads with the consideration of how much time this other path might take. Those most likely to be harmed are individuals whose roads and paths will become used because in this algorithm it might value paths that might not actually be great to travel on. Thus certain neighborhoods or places of travel might be overwhelmed as the algorithm does not account for the capacity of the road. I would be willing to implement it if there were additional checks that accounted for things like speed limit, size of road, residential area so that it doesn't make communities overwhelmed with traffic.

- Correctness Assessment: [Does the code compile? Does the code run without errors? Does the code output the correct answer on test cases? What edge cases have you tried? What issues are you aware of?]

The code compiles without giving errors. The output gives what seems to be the best option compared to walking on google earth and the provided map system. At some point I had the output giving out a path of a bit larger size so I think it is now the most accurate path.

- Modularity/Extensibility/Flexibility: [Describe how you broke up your code into methods or classes in order to make the code easier to organize/reuse. Is there anything that you hard-coded that would present challenges if the input were modified, or if additional functionality was required?]

In most cases my code is not hard coded and could be implemented if you changed the start and end point. I tested this by using other points and seeing if they did in fact create what seemed to be the shortest path, I think if there were things like negative cycles there would need to be loop checkers in place. This program only works for roads and the data types would have to change if thinking about other types of distances. Outside of this though the code should be able to be reused as it is broken up into the two methods and then the individual parts of each method.

- Readability: [Is your code commented? Did you choose variable and method names that help the reader understand their purpose?]

My code is commented, I tried to comment parts into sections but I think I also had the assumption that the person looking at it would somewhat already understand Bellman-Ford. Most of my variable names came from the original algorithm (like u and v). I used a lot of simple variables like P to represent the distance path and then P1 to keep track of the roads themselves. I also reused variables across my two methods, for example my variable ulist was used twice and represented the same thing each time it was used.

- Effort: [Describe the effort (time, approaches you took when stuck, resources you used including people you worked with) that you put into the assignment. If you could go back and do it again, how would your approach change?]

I attempted to write a significant amount of code for this about four times, on four different days. I think the first time was to outline what I needed to do and to try to figure out what the given code did. The next few days were just trying to figure out my bugs and errors, which took a lot longer than I thought it would. If I could go back I think I would spend more time outlining what my code should do because I think I tried writing code too soon and just trying to figure out how

the data types work. I also think it would have helped if I had talked to people about the code, as I think a lot of my errors were small but came out of me just not discussing the problem with others.