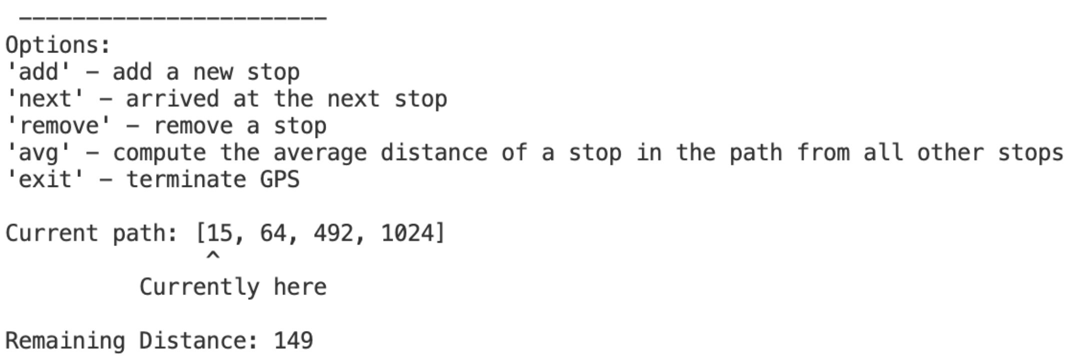
https://github.com/beaubeaubeaubeau/ds210Final

My final project is a basic GPS. It is divided into separate modules: main.rs (containing the main program), functions.rs (containing all auxiliary functions), and graph.rs (containing implementation for the directed graph). The graph which I will be working with is roadNet-TX.txt, which holds information about the connectivity of all the roads in Texas.

main.rs starts by reading the .txt file and converting it into an undirected graph. It then initializes a new VecDeque called 'path', which will hold the path of nodes in our GPS. Finally, it will loop an unspecified number of times until the user terminates the program themselves.

The loop presents a list of options with which the user can input: 'add', 'next', 'remove', 'avg', and 'exit'. It will also show the current 'path' queue and call *check_distance()* for the total distance between each node in the path (starting from the first node down to the final node). The distance computation makes use of the breadth-first search algorithm, which loops through an entire undirected graph and can return the shortest path to a specified node. While 'path' has a length of 2 or more (otherwise it simply returns 0), *check_distance()* loop through the 'path' vecdeque, summing the distance between each node and the next until it reaches the end of 'path'.

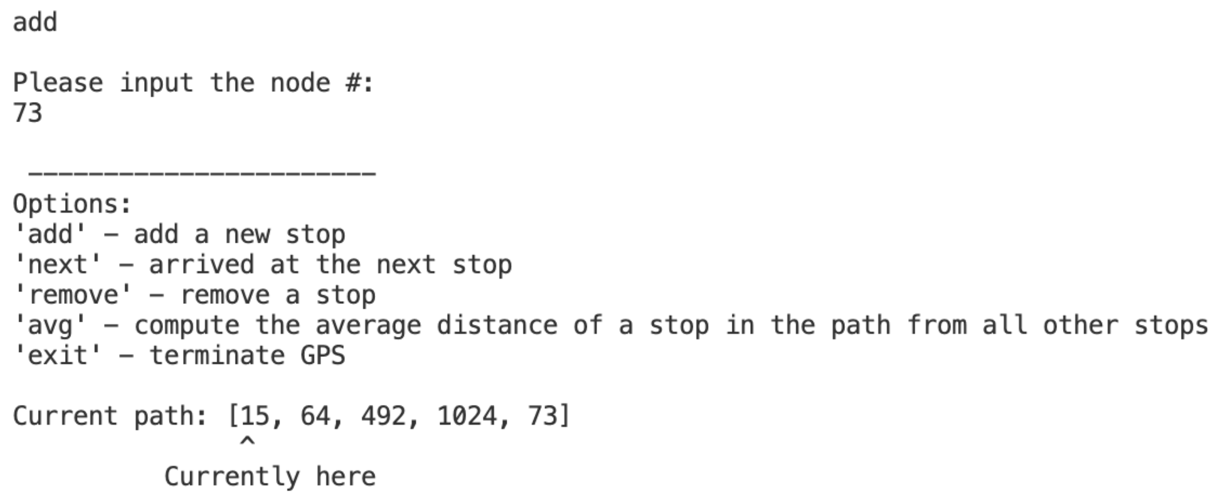
Ex. Display of options, path, and distance



```
 -----------------------
Options:
'add' - add a new stop
'next' - arrived at the next stop
'remove' - remove a stop
'avg' - compute the average distance of a stop in the path from all other stops
'exit' - terminate GPS

Current path: [15, 64, 492, 1024]
               ^
          Currently here

Remaining Distance: 149
```

'add' would call the *add_stop()* function, which reads a node from the user and checks whether the input is valid or not (negative input would trigger panic due to type constraint of usize and an input too high would return an Err). The program would then match the return value to an Ok(result) or an Err(error). The former would add the node to the end of 'path' while the latter would print "Input is not valid". Similar node checking is performed for 'remove' and 'avg'.

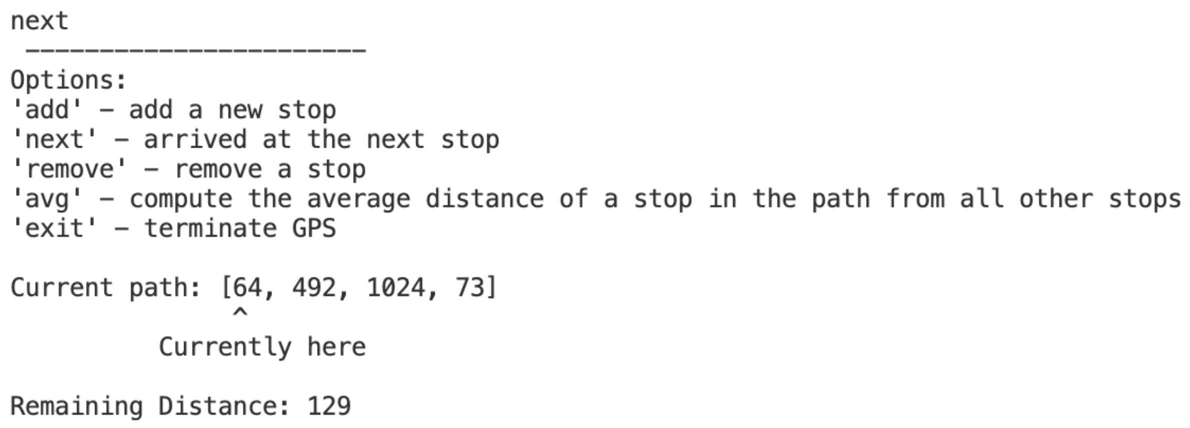
Ex. ‘add’ being inputted and a node being read



```
add

Please input the node #:
73

 -----------------------
Options:
'add' - add a new stop
'next' - arrived at the next stop
'remove' - remove a stop
'avg' - compute the average distance of a stop in the path from all other stops
'exit' - terminate GPS

Current path: [15, 64, 492, 1024, 73]
               ^
          Currently here
```

‘next’ simulates the arrival at the next stop for the GPS. ‘next’ simply calls *pop_front*() on ‘path’, removing the first value (or nothing if the path is empty).

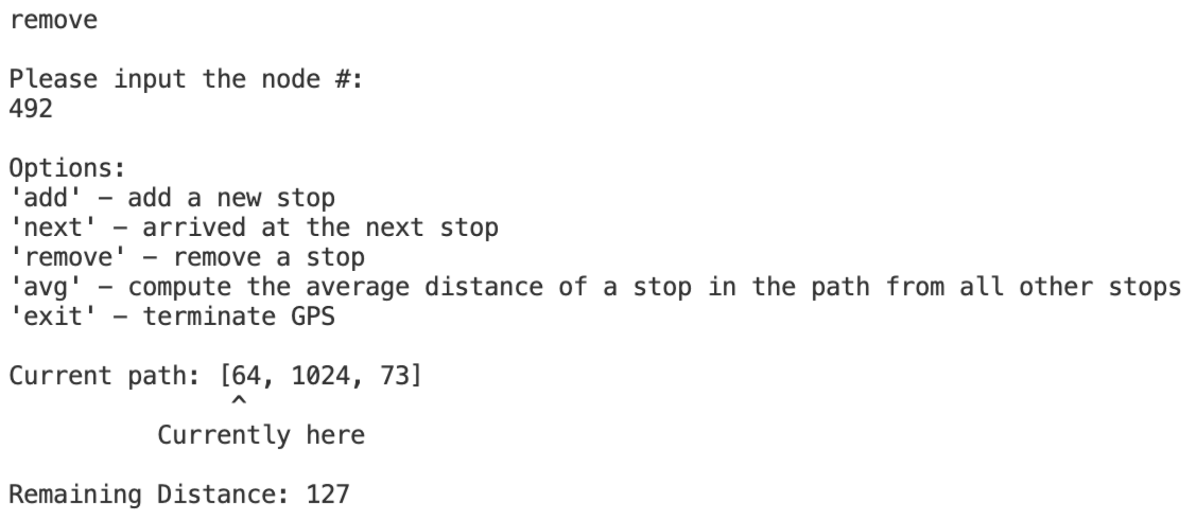
Ex. ‘next’ being inputted



```
next
 -----------------------
Options:
'add' - add a new stop
'next' - arrived at the next stop
'remove' - remove a stop
'avg' - compute the average distance of a stop in the path from all other stops
'exit' - terminate GPS

Current path: [64, 492, 1024, 73]
               ^
          Currently here

Remaining Distance: 129
```

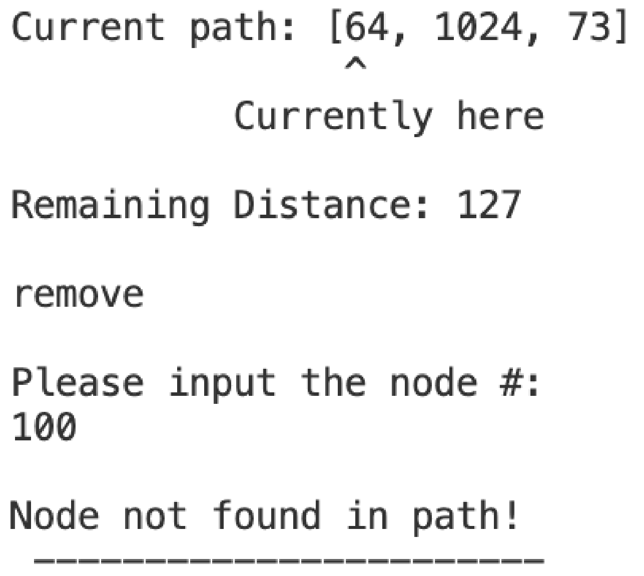
‘remove’ calls the *remove_stop()* function, which asks for a node and does some basic input validation, similarly to the *add_stop()* function. However, *remove_stop()* also loops through the VecDeque and removes the node that matches the inputted node. If such a node is not found in the path, *remove_stop()* returns an Err saying that the node was not found in the path.

Ex. ‘remove’ being inputted with node 492



```
remove

Please input the node #:
492

Options:
'add' - add a new stop
'next' - arrived at the next stop
'remove' - remove a stop
'avg' - compute the average distance of a stop in the path from all other stops
'exit' - terminate GPS

Current path: [64, 1024, 73]
               ^
          Currently here

Remaining Distance: 127
```

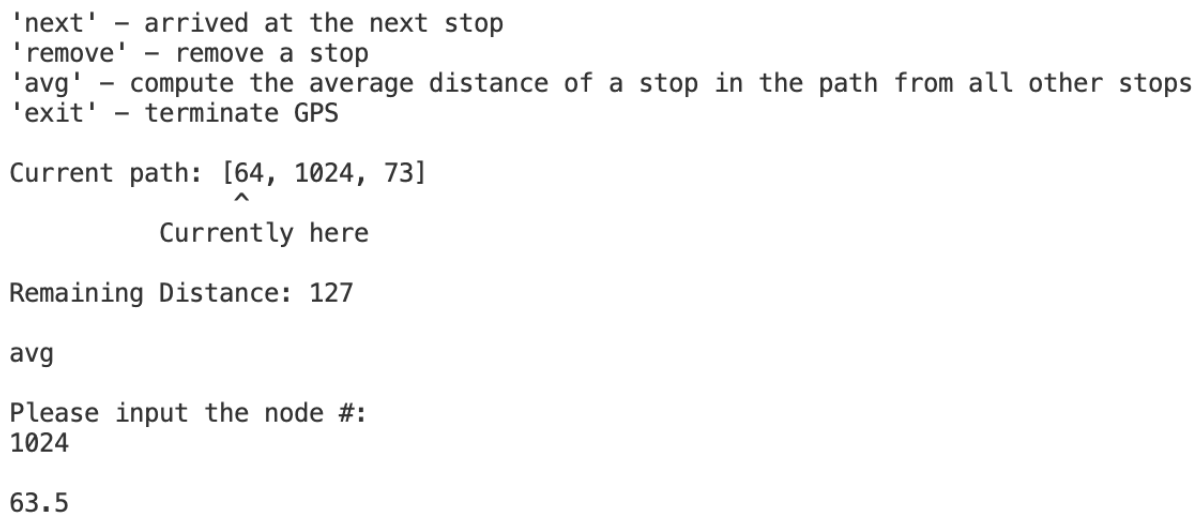
Ex. ‘remove’ being inputted with node 100



```
Current path: [64, 1024, 73]
               ^
          Currently here

Remaining Distance: 127

remove

Please input the node #:
100

Node not found in path!
 ----------------------
```

‘avg’ asks for a node. It then loops through the path and calls breadth-first search between that node and all other nodes in the path, summing each distance. Finally, it returns the average of the distances of the nodes in the path from the inputted node. This provides an idea for the centrality of a node in relation to the other nodes in the path.

Ex. Inputting ‘avg’ and then 1024



```
'next' - arrived at the next stop
'remove' - remove a stop
'avg' - compute the average distance of a stop in the path from all other stops
'exit' - terminate GPS

Current path: [64, 1024, 73]
               ^
          Currently here

Remaining Distance: 127

avg

Please input the node #:
1024

63.5
```

Finally, ‘exit’ would use the *break* keyword to terminate the program and exit out of the loop, ending the GPS.

Typing any option not provided by the options list would call *continue* and trigger another iteration of the loop.