Data: http://konect.cc/networks/subelj_euroroad/

This navigational algorithm takes in road map data, where each node represents a city and an edge represents the roads that connect them. The goal of this algorithm is to use breadth-first-search (BFS) to help the user find the shortest path from one city to another within the given network of cities and roads. The code is split into three modules: dataset.rs, bfs.rs, main.rs.

dataset.rs
The first module contains two functions: “read_file” and “create_adjacency_list”.

The first function, “read_file”, loads in undirected map data, opens it, and uses BufReader to read and parse through the file line by line. Each line is trimmed into a vector of string slices, then the whitespaces are used as separators. The results of the function are tuples that each contain a vertex count and a vector of a tuple that contains a label and edge value.

“create_adjacency_list” takes the results of “read_file” and creates a HashMap where each node is a key and every connected neighboring node is a value. It uses a for loop to iterate through each of the tuples from the output of “read_file” and pushes the corresponding values to the HashMap.

bfs.rs
The second module contains two functions: “bfs” and “find_route_with_waypoint”.

The “bfs” function is an implementation of the breadth-first-search (BFS) algorithm, and takes in the HashMap created in “create_adjacency_list”, a start node, and a goal node or the destination node, and outputs an optional value that can be a vector holding a usize value, or None.

To give a brief description of both the code and the BFS algorithm in general, it starts with the initialization of a queue for neighboring nodes to be processed, and a HashSet that holds all the visited nodes. It will then enqueue all the unvisited neighbors of the start vertex, process them, then add them to the HashSet, then repeat. Once the goal vertex is processed as well as all reachable nodes, the algorithm will then backtrace its path back to the start vertex to outline the “shortest path”.

The “find_route_with_waypoint” function takes in the adjacency list and the start, waypoint, and goal vertices. The function calls “bfs” two times, one for the start vertex to the waypoint vertex, and another for the waypoint vertex to the goal vertex. Instead of calling “bfs” twice in the main function, this function is called once.

main.rs
The third module contains four functions, two of which are tests: read_user_input, main, no_path_test, and specific_path_test.

“read_user_input” is a simple function that takes in user input in the datatype of a string, checks if it is valid, then converts it into a usize value. It is called three times in the main function for the start, waypoint, and goal vertices.

“main” is where all the aforementioned functions connect to run the entire algorithm. It starts by opening the file using “read_file”, then accepts user input through “read_user_input”, and finally runs the BFS algorithm through “find_route_with_waypoint”.

“no_path_test” is a test function that checks whether or not the “bfs” function works properly when given a graph with disconnected nodes. I created a hypothetical graph with given start and goal vertices, let it run, and compared it with the expected path.

“specific_path_test” is a test function that also checks the “bfs” function but whether or not it gives the proper path. I created a hypothetical graph with given start and goal vertices, let it run, and compared it with the expected path.

Output
The output is interactive with the user as mentioned above, but eventually outputs a path if it exists, or an error message.

The following are example situations with hypothetical inputs: