Assignment 2 Review

To complete this assignment, I made a map (Map<String, String> realNames) to store the names of places for each letter provided in the places.txt file, an array list of type *Constraints* that keeps the string values of start and end node, as well as the probability of the constraint to occur. These constraints are included upon the creation of the graph using a random variable, by comparing it to the specified constraint probability if the letters of both start and end node match. The hardest part of the task was figuring out how to make the graphs. Initially, I thought about using a matrix to store the data, but after consultations with some of my colleagues, I decided to make the graphs using classes which contain hashmaps and names for nodes. This way, the graphs function similarly as if it was created with a matrix, but it is much easier to add new nodes, as well as to include constraints between the specified nodes. The graph is made using two classes, *Graph* and *Nodes* class, where the *Graph* class contains a map of String and Nodes pairs, to store letters representing the nodes as keys and Nodes themselves as values. The Nodes class contains *String name* and *Map<Nodes, Integer> neighbors* as its attributes, in order to store the names of neighbor nodes, as well as to store the nodes and distances of neighbors in a map. After making the graphs with included constraints, I created a class called Dijkstra which has the *getShortestPath* method to find the shortest path between all nodes in a graph. This method also takes into consideration different cases where nodes are not connected (returns -1), where nodes do not exists or the graph is empty (returns -1), as well as when you try to find the shortest path between the same node (returns 0). In order for this method to work, I made another class called *NodesAndDistances* in order to keep track of distances between nodes when determining the shortest path. Results of all of the graphs were written onto separate .txt files, showing the shortest paths between each of the nodes (if the nodes are connected, if not it shows -1). I made seven separate tests to check for all possible cases. All tests were done with an empty *Constraints* arraylist except for one, where I put the probability of constraints to be 100% in order to check if the constraints will be applied. Other tests include a test when all nodes are connected, a test where the source and/or destination node does not exist, test between two nodes connected with one another, test for when you try to find the shortest path between the same node, a test for when the graph is empty, and a test for a sample graph which was generated using ChatGPT.