Introduction

Our dataset models the road networks of Pennsylvania and was chosen from the Stanford Large Dataset Collection. In this dataset, intersections and endpoints are represented by nodes while the roads connecting these intersections or endpoints are represented by undirected edges. In our final output, we accounted for the first 250 nodes (node 0 to node 249).

Our goal for this project was to create a program which will implement a DFS traversal, and the Dijkstra's and Landmark Shortest Path algorithms on our dataset. We successfully implemented Dijkstra's algorithm and the Landmark Shortest Path algorithm on our dataset. However, during project development we opted to implement a BFS traversal, instead of a DFS traversal, as it is more appropriate for our dataset.

Code Base

Retrieving Our Data

Development/Approach: The original dataset includes 1088092 nodes and 1541898 edges, but for our project, a smaller dataset of the road networks of PA were chosen to work with. The readFile class was established for this purpose. The dataset file "roadNet-PA.txt" was converted to a vector of strings. This vector of strings was then modified to an undirected vector containing pairs of nodes such that no two pairs of nodes are repeated. The dataset was then limited to include only the Nodes 0-249, making our project more easily testable.
Outcomes: We were able to read in a relatively large dataset of directed nodes and edges from a .txt file, and modify it to only include undirected nodes and edges (by removing repetitive edges). We also limited the range of the dataset to only the first 250 points. This allowed us to more easily work with the road network dataset when implementing BFS, Dijkstra's, and Landmark algorithms. However, it also introduced some challenges as only leaving the first 250 nodes meant there could be disconnections within the graph.

Graph Class

Development/Approach: We derived the basis of our Graph class from the Graph class which was provided to us in lab_ml. After cleaning the code to remove methods we didn't need to implement for our project, we then added the necessary helper functions and modified the respective data types to be able to better implement this class in our project. The purpose of each method in this class is essentially as the name suggests.

BFS Traversal

Development/Approach: The BFS class was created to implement a breadth-first search algorithm traversing the graph of our road network. This class stores vectors of edges that are unexplored edges, discovered edges, and cross edges. It also keeps track of visited vertices and visited edges in two maps and stores the traversed nodes in the private variable output_queue.
Outcomes: We were able to successfully implement a breadth-first search algorithm that writes out the breadth-first searched vertices visited in order and the visit state of each edge in the file "BFS.txt". Click here to view the output of the BFS Traversal

Dijkstra's Algorithm

Development/Approach: To implement our Dijkstra's Algorithm, we used the typical logic/conditions. However, rather than the approach of using a Priority Queue, we used three vectors to keep track of the total distances, vertices which we have visited, and the pathway which we take. Essentially, we iterate through every vertex, and each time we examine the adjacent vertices and select the one with the smallest edge weight. We then update our vectors and continue this process until all vertices are processed. The end result is a distance vector with all distance to the source vertex, a prevStep vector with essentially an uptree that eventually leads back to the "root" (source). We then set the current graph object's vector variable Ddistance and DprevStep to this end result to be used with the Landmark Algorithm.
Outcomes: We were able to successfully implement the Dijistrak's Algorithm; however, we did come across one limitation in out implementation. Currently, since we only consider some of the nodes from our dataset, the portion of the graph which is connected to nodes outside of the region we consider is disconnected, and cannot implement the Dijkstra's Algorithm, which is why we in turn cannot implement the Landmark Algorithm on disconnected nodes as well. If we were to continue working on this project in the future, we would like to include an additional helper function which filters out the disconnected portions of the graph, or, eventually even be able to manage this program for all the nodes given in the dataset. Click here to view the output of the Dijkstra's Algorithm

Landmark Shortest Path Algorithm
Development/Approach: Our approach towards solving the Landmark problem was to implement Djikstra's algorithm from the landmark vertex as the source. Doing this we had the distances and the two shortest paths from the source and destination to the landmark.
Outcomes: We were able to successfully solve the landmark path problem by using the information collected from the Dijkstra's algorithm and save it as the graph object's public variable; however, we have identified that there are alternative approaches towards solving the Landmark path problem. This includes using a DFS method to find all the paths in between the given node A and node B, filtering out the paths which include the Landmark node C, calculating the distance of each of these paths, and then returning the smallest one. We understand that this is a brute force approach, but it is one of the many ways. On Click here to view the output of the Landmark Algorithm

Conclusion
Within the certain limitations discussed in our final presentation, we were basically able to meet our project goals. This project allowed us to use our knowledge and skills from CS225 towards creating something substantial, based on the real world. Apart from implementing the actual algorithms, we were also given the opportunity to demonstrate more practical skills for the workplace, such as retrieving and cleaning data, creating our own makefiles and test cases, and having tangible outputs outside of test cases. We hope to continue working on this project in the future, and using the skills and knowledge from future coursework and projects, we aim to overcome our current limitations and present our program to many other respective data sets.