COMPUTATION ANALYSIS

WITH

PATH FINDING ALGORITHMS

TABLE OF CONTENTS

1. Dijkstra Algorithm

================================================================================

* Elapsed time
* Shortest Path
* Time Complexity
* Open Street Map Dataset

1. Floyd War shall Algorithm

=================================================================================================

* Elapsed time
* Shortest Path
* Time Complexity
* Open Street Map Dataset

1. Brute Force Algorithm

=================================================================================================

* Elapsed time
* Shortest Path
* Time Complexity
* Open Street Map Dataset

Dijkstra Algorithm

Dijkstra's algorithm relies on the principle that once a node is marked as visited, the shortest path to that node has been found. Therefore, it ensures that the shortest path to each node is found progressively, building on previously found shortest paths.

initialization: Start with a source node and assign a tentative distance value to every other node, marking them as unvisited. Set the distance to the source node as 0 and all other nodes' distances as infinity initially.

Selecting the Closest Node: Among the unvisited nodes, choose the one with the smallest tentative distance. Initially, this will be the source node.

Update Neighbors' Distances: For the selected node, calculate the distance to each of its unvisited neighbors by adding the weight of the edge connecting them to the tentative distance of the current node. If this distance is less than the current tentative distance of the neighbor, update the neighbor's tentative distance.

Mark Node as Visited: Once all the neighbors of the current node have been examined, mark the current node as visited.

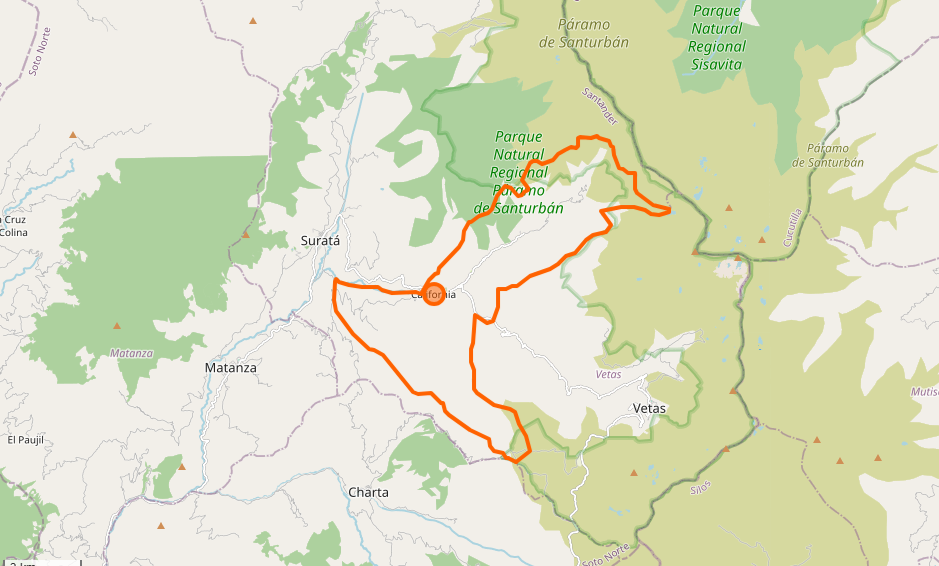
Repeat: Repeat steps 2-4 until all nodes have been visited. At each step, select the unvisited node with the smallest tentative distance, update its neighbors' distances, and mark it as visited.

Termination: When all nodes have been visited, the algorithm terminates. The final distances from the source node to all other nodes represent the shortest paths.

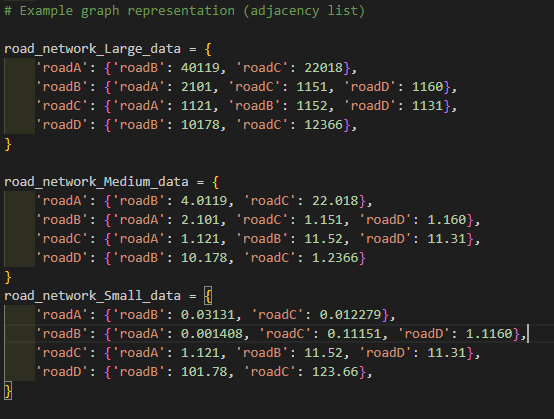
Dijkstra's algorithm works only for graphs with non-negative edge weights. If the graph contains negative edge weights, a different algorithm like the Bellman-Ford algorithm is needed.

Open Street Map Real World Data

(California)



Translated Graph Dataset

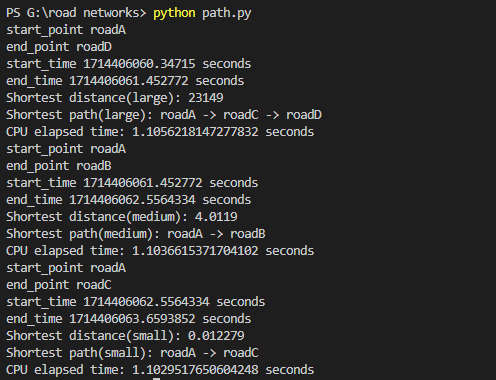


Shortest Path among Road Networks

We used the open street map data of road networks as a raw reference. The idea was to used that raw data as an input to path finding algorithm. So, we structured the entire data as graph in each vertex or node connects with an edge and every corresponding vertex would have a connection or neighbors’ previous edges. This interpretation of using such kind of distributive algorithm was to keep track of the previous vertices which could result in a very optimized traversal through the entire graph.

Time Complexity 0 (V +E) Best Case

PERFORMANCE ANALYSIS



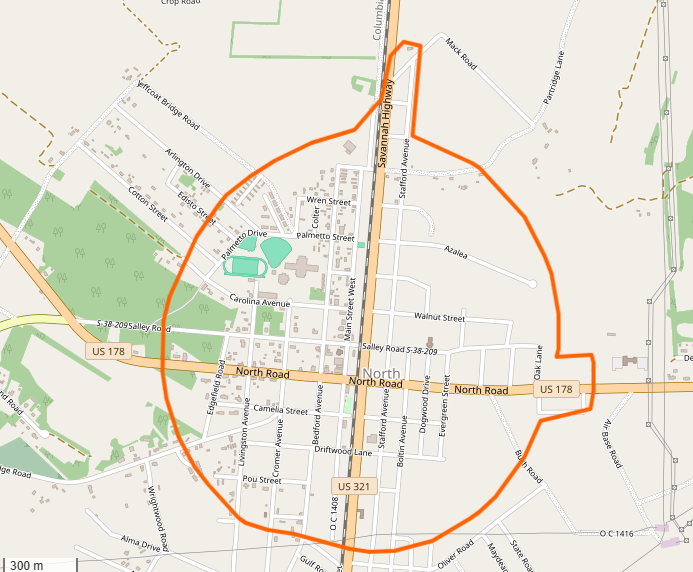
Floyd War shall Algorithm

The Floyd-War shall algorithm is another method for finding the shortest paths between all pairs of vertices in a weighted graph. Unlike Dijkstra's algorithm, which finds the shortest path from one source node to all other nodes, Floyd-War shall finds the shortest paths between every pair of nodes in the graph. Floyd-War shall algorithm is notable for its simplicity and efficiency in finding the shortest paths between all pairs of vertices in a graph. However, its time complexity is, where is the number of vertices in the graph, making it less efficient than Dijkstra's algorithm for finding the shortest path from one source node to all other nodes in large graphs. However, its advantage lies in its ability to handle graphs with negative edge weights as long as there are no negative cycles.

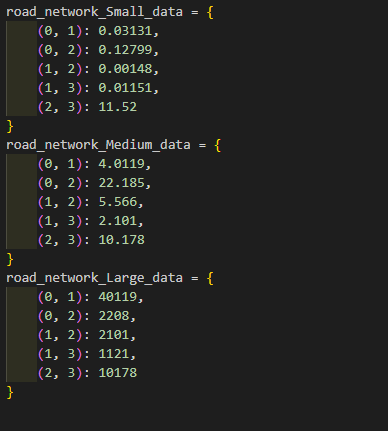
Time Complexity 0(V^3) Average Case/Worst Case

OPEN STREET MAP RAW DATA

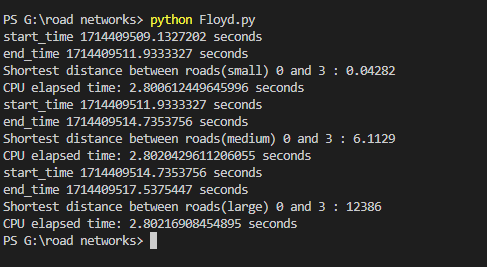
(North America )



Translated Adjacency List



Performance Analysis



Brute Force Algorithm

Brute force pathfinding is a straightforward approach to finding the shortest path between two points in a graph by exhaustively searching through all possible paths. While this method guarantees finding the shortest path, it is often impractical for large graphs due to its exponential time complexity.

Here's a step-by-step explanation of how brute force pathfinding works:

Generate all Possible Paths: Start by generating all possible paths from the starting node to the destination node. This can be done recursively or iteratively, exploring all possible combinations of edges until reaching the destination node.

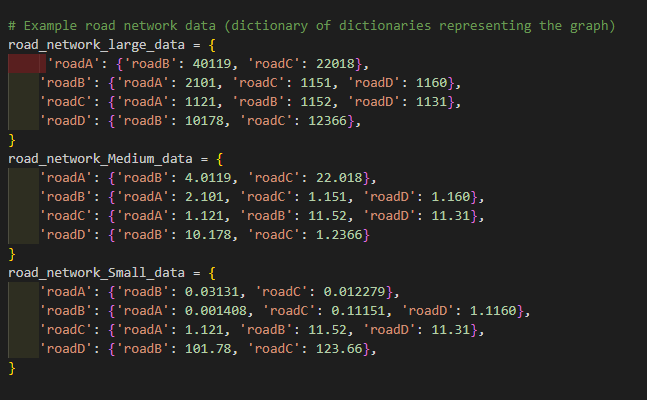
Calculate Path Lengths: For each generated path, calculate its length by summing the weights of its constituent edges.

Find the Shortest Path: Compare the lengths of all generated paths and identify the one with the shortest length. This path represents the shortest path between the starting and destination nodes.

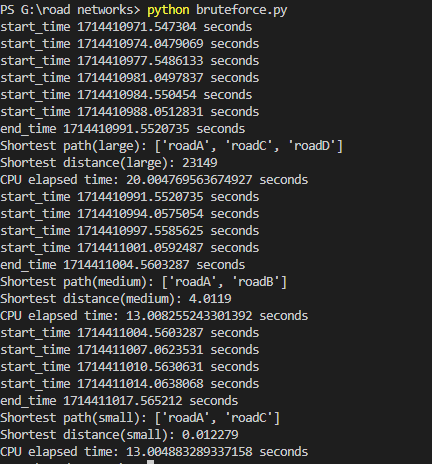
Output: Once the shortest path is found, output it as the result of the pathfinding process.

Time Complexity 0(N^2) Worst Case

We apply the same California Translated Graph Dataset:



Performance Analysis



CONCLUSION

So after applying all the algorithms the over all performance analysis show that Dijkstra Algorithm is the fastest solution when applied to a same data set due to its corresponding neighbors and vertices checking property we can calculate the shortest path in less time while consuming less system resources.