13.8 Project 3 Smart Delivery Route Planner- Design

Project 3 Group 13: Kurtis Mok, Andre Jiang, Sydni Yang

Date: May 24, 2025

1. Overview

Purpose: The purpose of this project is to create a logistics system through the design of a Smart Delivery Route Planner which determines the best and most efficient route to choose across a city. We hope to optimize time and fuel efficiency for our client's delivery drivers as they deliver to different customer locations. Our project will use graphs and algorithms in order to achieve this.

Inputs: Our program will accept input for depot and delivery locations as well as any additional delivery stops along the way. Furthermore, it may also take a .csv file for multiple different depot and delivery locations.

Output: The Smart Delivery Route Planner aims to output the most efficient path to complete all of the deliveries. It will output a step-by-step delivery plan showing each stop and its distance. It will then compute the total distance or time required to complete the delivery (including the stops)

Worked Out Example:

Enter depot location: PCC

Enter delivery stops (comma separated): Canada, Hawaii, Hollywood

Delivery Plan:

1. PCC -> Hollywood (24 km)

2. Hollywood -> Canada (3,292 km)

3. Canada -> Hawaii (4,345 km)

Total distance: 7661 km

2. Function Designs (Follow CS034 Section 5.6 format)

1. `build_graph(filename)`

Purpose: The purpose of this function is to read a file that represents city road names and build a graph where nodes represent intersections and edges (with distance/time) represent roads.

Parameters/Return Values:

- Parameter(s): filename (string) - path to the file containing all the data and information on the city roads.
- Return value(s):

Pseudocode:

Open and read the file

For each file line:

    Analyze the starting and ending nodes, and the weight

    Add connections to the graph if both directions are not directed yet

Return the new graph

2. `is_route_possible(graph, start, end)`

Purpose: The purpose of the function is to check if the path or route is available between the two nodes using various search algorithms.

Parameters/Return Values:

- Parameter(s): graph (dictionary) - represents the graph, start (string) - starting node, end (string) - ending node
- Return value(s): True - route is possible, False - route is not possible

Pseudocode:

Initialize a stack with the starting node

Maintain a set of the nodes that were already visited

While the stack is not empty:

    Pop the current node

    Return True if the current node is the end node

    Put all unvisited neighbors in the stack

Return False

3. `find_shortest_path(graph, start, end)`

Purpose: Find the shortest path between two nodes, considering edge weights

Parameters/Return Values:

- Parameter(s): graph (dictionary) - weighted graph representation, start (string) - starting node, end (string) - destination node
- Return value(s): tuple (path, distance) - path as list of nodes and total distance, or (None, float('inf')) if no path exists

Pseudocode:

Initialize the priority queue

Initialize distances dictionary with all nodes set to infinity, start set to 0

Initialize visited set

While priority queue is not empty:

Pop node with minimum distance

If node is the end node:

Return the path and distance

If node already visited:

Continue to next iteration

Mark node as visited

For each neighbor of current node:

Calculate new distance through current node

If new distance is shorter than recorded distance:

Update distance and add priority queue with update path

Return None if no path is found

4. `plan_delivery(graph, depot, deliveries)`

Purpose: Plan an efficient delivery route that visits all delivery locations and returns to the depot, minimizing total travel distance

Parameters/Return Values:

- Parameter(s): graph (dictionary) - weighted graph of road network, depot (string) - starting/ending location, deliveries (list) - list of delivery location nodes
- Return value(s): tuple (route, total_distance) - optimal route as list of nodes and total distance traveled

Pseudocode:

Initialize a list of all locations to visit, starting and ending with the depot

Initialize an empty list for the best route

Initialize the minimum distance as infinity

For each possible order of the delivery locations:

    Create a route starting at the depot, followed by the delivery order, and ending at the depot

    Set total distance to 0

    For each pair of consecutive locations in the route:

        Add the distance between them to the total distance

    If the total distance is less than the minimum distance:

        Update the minimum distance

        Update the best route

Return the best route and the minimum distance