Isaiah Lugo

CSM IV – Algorithms

Final Project Report: Delivery Optimization for Al-GO-rithms

Week 14

**Overview**

The delivery company **Al-GO-rithms** hired me to optimize their logistics network to meet the challenges of the busiest time of the year. Their delivery network spans various locations (nodes) interconnected by roads (edges), where each road has a travel cost (weight) associated with it. The objective was to design three key algorithms to solve different logistical optimization challenges. These algorithms are:

1. **Algorithm 1: Lowest Cost Delivery Between Two Locations**
2. **Algorithm 2: Best Path from the Hub**
3. **Algorithm 3: Dynamic Network Changes**

Each algorithm has a specific real-life application and uses graph-based approaches to solve delivery optimization problems.

**Algorithm 1: Lowest Cost Delivery Between Two Locations**

**Problem Statement**

Determine the shortest path (lowest total cost) between a starting node and an ending node.

**Solution**

This problem is solved using **Dijkstra’s Algorithm**, which finds the shortest path in a weighted graph with non-negative weights.

**Function Inputs**

* A weighted graph (dictionary)
* A start node
* An end node

**Output**

* The shortest path as a list
* The total cost of that path

**Example Test Case**

algorithm\_1(example\_graph, "A", "E")

# Output: (['A', 'C', 'B', 'D', 'E'], 10)

**Algorithm 2: Best Path from the Hub**

**Problem Statement**

Find the most efficient path to connect all nodes in the graph starting from a central hub, with the goal of minimizing total cost.

**Solution**

This is a classic **Minimum Spanning Tree (MST)** problem. The MST is constructed using **Prim’s Algorithm**, starting from the hub node.

**Function Inputs**

* A weighted graph (dictionary)
* A hub node

**Output**

* The edges of the MST
* The total cost

**Example Test Case**

algorithm\_2(example\_graph, "A")

# Output: ([('A', 'C', 2), ('C', 'B', 1), ('B', 'D', 5), ('D', 'E', 2)], 10)

**Algorithm 3: Dynamic Network Changes**

**Problem Statement**

Update the delivery network by adding or removing connections and find the new most efficient network configuration.

**Solution**

This builds on Algorithm 2 by modifying the graph dynamically:

* Remove specified edges.
* Add new edges.
* Re-run Prim’s Algorithm to generate the new MST.

**Function Inputs**

* A weighted graph (dictionary)
* A hub node
* A list of edge strings to remove
* A list of tuples representing new edges to add

**Output**

* Updated MST
* Total cost

**Example Test Case**

algorithm\_3(example\_graph, "A", ["C-E"], [("B", "E", 3)])

# Output: ([('A', 'C', 2), ('C', 'B', 1), ('B', 'E', 3), ('D', 'E', 2)], 8)

**Test Case Summary**

Each algorithm was tested using the sample graph provided. All functions returned correct outputs based on theoretical expectations. Additional edge cases such as disconnected nodes or empty graphs can be added for future validation.

**Challenges and Solutions**

* **Challenge:** Ensuring Dijkstra’s algorithm doesn't revisit nodes unnecessarily.
  + **Solution:** Used a priority queue and checked visited nodes.
* **Challenge:** Avoiding duplicate edges in Prim’s MST.
  + **Solution:** Checked if a node was already visited before adding to the MST.
* **Challenge:** Updating graph dynamically without data loss.
  + **Solution:** Carefully modified both directions of undirected graph edges.

**Conclusions and Future Improvements**

This project demonstrates how classical graph algorithms can solve real-world delivery logistics problems effectively. In the future, the following enhancements can be explored:

* Add support for real-time traffic updates or weights that change over time.
* Visualize the graph and delivery paths using a library like networkx or matplotlib.
* Implement parallel processing for larger datasets or networks.

A screenshot of a computer

AI-generated content may be incorrect.