Finding Shortest and Fastest Paths from Delhi to Mumbai, Chennai, and Kolkata

SUMIT KUMAR (230380720016)

Case Study: Finding Shortest and Fastest Paths from Delhi to Mumbai, Chennai, and Kolkata

1. Decomposition:

To tackle this problem, we can break it down into several steps:

1. Acquire the map data with distance and travel time information between cities.
2. Create a graph representation of the map.
3. Implement Dijkstra's algorithm to find the shortest path from Delhi to all other cities.
4. Modify Dijkstra's algorithm to consider travel time and find the fastest path from Delhi to all other cities.
5. Analyze and compare the results to determine the most efficient routes.

2. Pattern Recognition:

This case study involves finding the shortest and fastest paths, which indicates that it requires graph traversal algorithms. Specifically, we can use Dijkstra's algorithm for finding the shortest path and a modified version of Dijkstra's algorithm that considers travel time for finding the fastest path.

3. Abstraction:

We can abstract the problem as follows:

- Represent the map as a weighted undirected graph, where cities are nodes and the distances between cities are the edge weights.
- Implement two algorithms: one for finding the shortest path and the other for finding the fastest path.
- Analyze the results to compare the efficiency of different routes from Delhi to Mumbai, Chennai, and Kolkata.

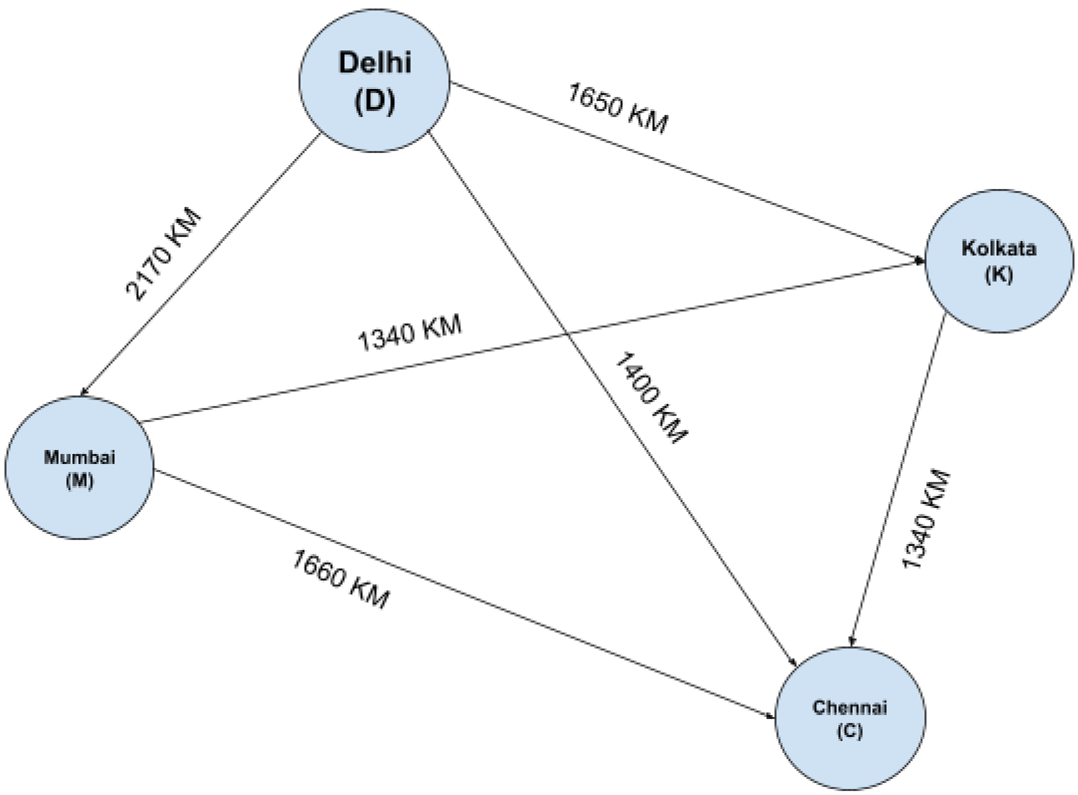
Finding Shortest and Fastest Paths from Delhi to Mumbai, Chennai, and Kolkata

SUMIT KUMAR (230380720016)

4. Algorithm:

Step 1: Create the Graph:

- Represent the map with cities (Delhi, Mumbai, Chennai, Kolkata) as nodes and the distances between them as weighted edges.
- Include the travel time (in hours) as a second weight for each edge.



Finding Shortest and Fastest Paths from Delhi to Mumbai, Chennai, and Kolkata

SUMIT KUMAR (230380720016)

Step 2: Dijkstra's Algorithm for Shortest Path:

- Initialize an empty priority queue (min-heap) to store nodes with their current minimum distance from Delhi.
- Create a distance array to store the minimum distances from Delhi to all other cities and set all distances to infinity except Delhi (set distance to itself as 0).
- Start from Delhi and add it to the priority queue.
- While the priority queue is not empty, do the following:
  - Extract the node with the minimum distance from the priority queue.
  - For each of its neighboring nodes (cities), calculate the total distance from Delhi through the current node.
  - If this distance is less than the current minimum distance stored for the neighboring node, update the distance and add the neighboring node to the priority queue.
- At the end, the distance array will contain the shortest path distances from Delhi to all other cities.

Step 3: Modified Dijkstra's Algorithm for Fastest Path:

- Repeat the same steps as in Step 2, but instead of using edge distances as weights, use travel times as weights for the edges.
- Keep track of the minimum travel time from Delhi to all other cities.