**Project Report:   
Shortest Path Algorithms on Sri Lankan Cities**

**Group Members  
  
Fc110531/Fc211004 – Ganindu Deshapriya**

**Fc**

**Fc**

CCS3052 - Advanced Data Structures and Algorithms - Assignment

**1. Introduction**

Finding the shortest path between locations is a fundamental problem in computer science, with applications in navigation systems, logistics, and network optimization. In this project, we applied and compared several graph algorithms to compute shortest paths between cities in Sri Lanka.

The dataset contained geographical coordinates of major Sri Lankan cities, from which a weighted graph was constructed. Distances were calculated using the Haversine formula, representing realistic great-circle distances. The project’s main objective was to evaluate the performance of classical shortest path algorithms and observe their suitability for real-world map-based applications.

**2. Algorithms Implemented**

1. **Dijkstra’s Algorithm**
   * Finds the shortest path from a source node to a destination.
   * Uses a priority queue for efficiency.
   * Guarantees optimal paths in non-negative weighted graphs.
2. **Bellman-Ford Algorithm**
   * Relaxes edges repeatedly.
   * Handles negative weights (not present in this dataset).
   * Slower than Dijkstra for large graphs.
3. **A\* Search Algorithm** 
   * Extension of Dijkstra with a heuristic (Haversine distance).
   * Guides the search towards the goal, improving efficiency.
   * Produces the same shortest path as Dijkstra but faster in practice.

**3. Methodology**

* **Dataset Preparation**
  + CSV file of Sri Lankan cities with latitude and longitude.
  + Graph constructed by connecting each city to its *k nearest neighbors* (k=10).
* **Distance Metric**
  + Haversine formula used to calculate edge weights (great-circle distance in km).
* **Evaluation**
  + Each algorithm was run on selected city pairs (e.g., Colombo → Kandy, Jaffna → Galle).
  + Metrics: path length, distance (km), and execution time (s).
  + Folium library used for map-based visualization of paths.

**4. Results  
  
(harima results tikaka SS damu methanata, test cases 3kata wage)**

| **Algorithm** | **Path Found** | **Distance (km)** | **Execution Time (s)** | **Notes** |
| --- | --- | --- | --- | --- |
| Dijkstra | ✅ | XX | YY | Reliable, slower than A\* |
| Bellman-Ford | ✅ | XX | YY | Correct but slower |
| A\* | ✅ | XX | YY | Fastest, same path as Dijkstra |

*(XX and YY will be replaced with actual results from test runs.)*

**5. Observations**