Table of Contents

I. Pathfinding and Search Formulation

Goal Formulation

Goal State: The user-specified goal city

Goal test: When expanding a node, check $current\_city == goal\_city$. If true, stop.

Problem Formulation

Initial State: The start city given by the user.

Actions: From a city, you can travel along any adjacent highway edge to a neighboring city.

Transition model: $Result(state, action)$ = move from current city to the adjacent cities via the chosen road segment.

Path cost:

- Unweighted (BFS/DFS): each action costs 1 hop.
- Weighted: action cost equals edge miles; path cost is the sum of miles.

II. Frontier type per algorithm; explored set; node vs state

BFS	DFS	UCS (Dijkstra)	Greedy	A*
Frontier: FIFO queue	Frontier: stack(LIFO)	Frontier: min-priority queue	Frontier: min-priority queue keyed by h(n)	Frontier: min-priority queue
Explored Sets: keeps cities already removed from the queue	Explored Sets: keeps cities already removed from the queue	Explored Sets: $closed\ set + best\_g[s$ (lowest known cost)	Explored Set: cities popped from the queue	Explored Set: $closed\ set + best$
Node vs State: Node = (city, parent, depth) State = city name	Node vs State: Node = (city, parent, depth) State = city	Node vs State: Node = (city, parent, g) State = city	Node vs State: Node = (city, parent, h) State = city	Node vs State: Node = (city, parent, f) State = city

III. Graph representation choice of data structure and justification

Choice: Adjacency list implemented as a dictionary of lists.

Justification:

- Efficient neighbor expansion:
- Memory efficient for sparse graphs: Highway networks are sparse (each city connects to only a few others). Adjacency lists only store existing edges; no wasted space like in an adjacency matrix.
- Readable and flexible: lists store both neighbor name and edge weight, so you can use the same structure for unweighted (hops) and weighted (miles) algorithms
- Scales well: Space = O(V + E)