### **Notation / helpers**

* Start, Goal — city names
* Stops — set of required stopover city names (size k)
* neighbors(city) → list of (neighborCity, road\_distance, jeep\_flag) from Connections.csv and TrackType.csv
* H[a][b] — aerial (straight-line) distance between cities a and b from heuristics.csv (admissible)
* Represent visited as a bitmask integer of length k (map each stop to a bit index) for compact keys and fast subset operations
* MST\_cost(nodes) — MST cost over nodes using H as edge weights (Prim’s algorithm). Memoize by frozen subset key (bitmask)

### **High-level idea**

Search in state-space of (city, visitedMask) where visitedMask marks which required stops have already been visited. Goal reached when city == Goal AND visitedMask == allStopsMask.

Use A\* with f = g + h where:

* g = accumulated road distance so far (primary objective).
* h = admissible lower bound = minStart + MST\_R + minEnd (defined below). If you use lexicographic costs (distance, jeep\_count), use h = (distance\_lower\_bound, 0).

**Pseudocode:-**

FUNCTION Astar\_with\_stopovers(Start, Goal, Stops):

# Precompute mapping stop->bit index

stop\_list = list(Stops) # length k

stop\_index = {stop\_list[i] : i for i in range(k)}

allStopsMask = (1 << k) - 1 # mask with k ones

# Memoization for MST costs on subsets (key = bitmask)

MST\_memo = dict() # MST\_memo[mask] -> MST\_cost over stops in mask using H

# Helper: compute MST lower bound for stops represented by mask

FUNCTION MST\_lower(mask):

if mask == 0: return 0

if mask in MST\_memo: return MST\_memo[mask]

nodes = [ stop\_list[i] for i in range(k) if (mask >> i) & 1 ]

cost = Prim\_MST\_cost(nodes, H) # use H[u][v] as edge weights

MST\_memo[mask] = cost

return cost

# Heuristic: admissible lower bound for state (city, mask)

FUNCTION heuristic(city, mask):

remainingMask = mask

if remainingMask == 0:

return H[city][Goal] # aerial distance to goal

# min from current city to any remaining stop (aerial)

minStart = min( H[city][stop\_list[i]] for i in range(k) if (remainingMask>>i)&1 )

# MST cost across remaining stops (aerial)

mst = MST\_lower(remainingMask)

# min from any remaining stop to goal (aerial)

minEnd = min( H[stop\_list[i]][Goal] for i in range(k) if (remainingMask>>i)&1 )

return minStart + mst + minEnd

# A\* (distance only as primary objective)

openPQ = priority queue ordered by f = g + h

closed = dictionary mapping (city, mask) -> best\_g\_seen

startMask = 0

if Start in stop\_index:

startMask |= (1 << stop\_index[Start])

push into openPQ: node = { city: Start, mask: startMask, g: 0, path: [Start] }

with priority f = 0 + heuristic(Start, startMask)

while openPQ not empty:

node = pop lowest-f node

city, mask, g, path = node.city, node.mask, node.g, node.path

if city == Goal and mask == allStopsMask:

return path, g # found optimal route visiting all stops

# Skip if we already have a strictly better g for this state

if (city, mask) in closed and g > closed[(city, mask)]:

continue

closed[(city, mask)] = g

for (nbr, road\_dist, jeep\_flag) in neighbors(city):

newMask = mask

if nbr in stop\_index:

newMask |= (1 << stop\_index[nbr])

newG = g + road\_dist # if lexicographic cost: add to primary component

# optional: accumulate jeep\_count separately if needed

# Prune if we already have better

if (nbr, newMask) in closed and newG >= closed[(nbr, newMask)]:

continue

h = heuristic(nbr, newMask)

push into openPQ: { city: nbr, mask: newMask, g: newG, path: path + [nbr] }

with priority f = newG + h

return FAILURE # no feasible route that visits all stops