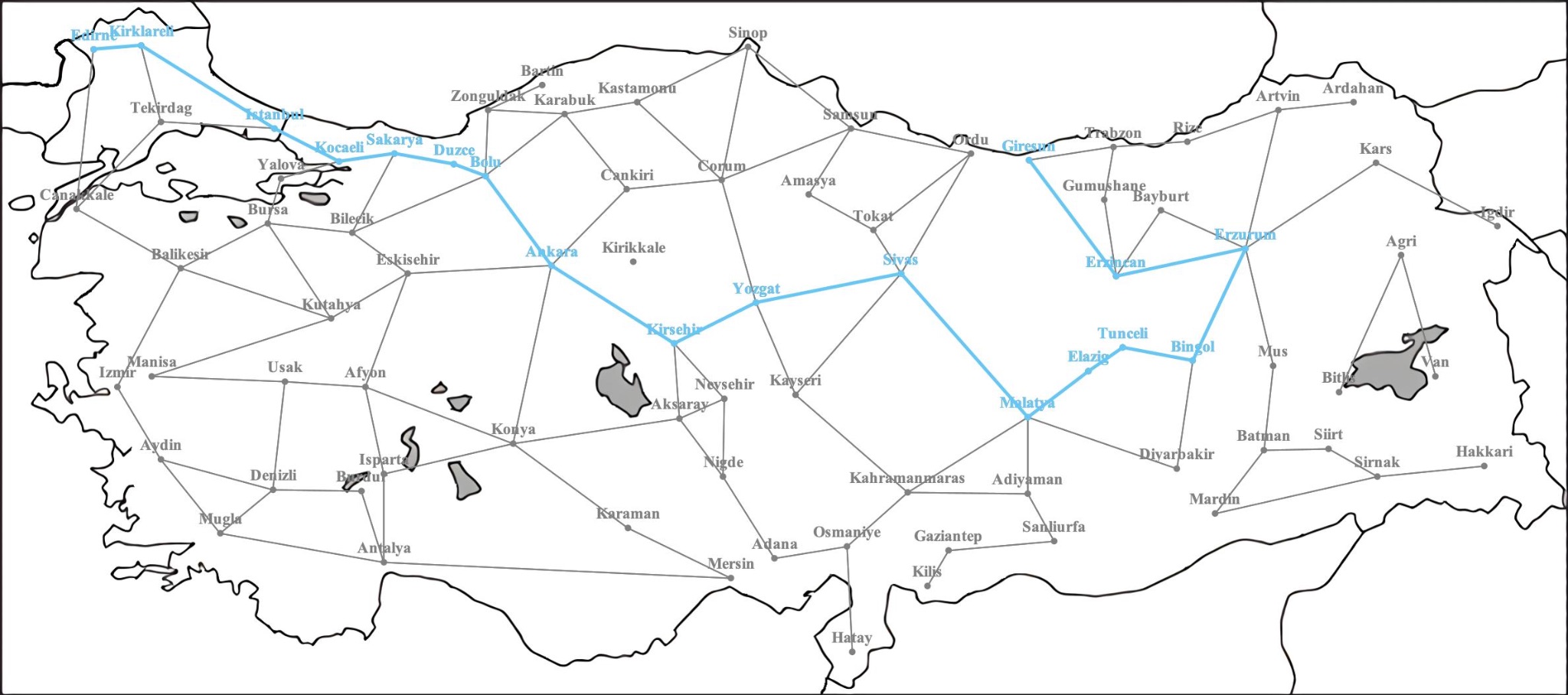
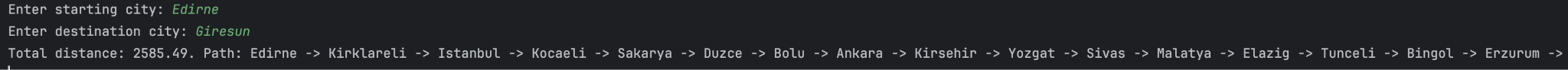
# **Turkey Navigation**

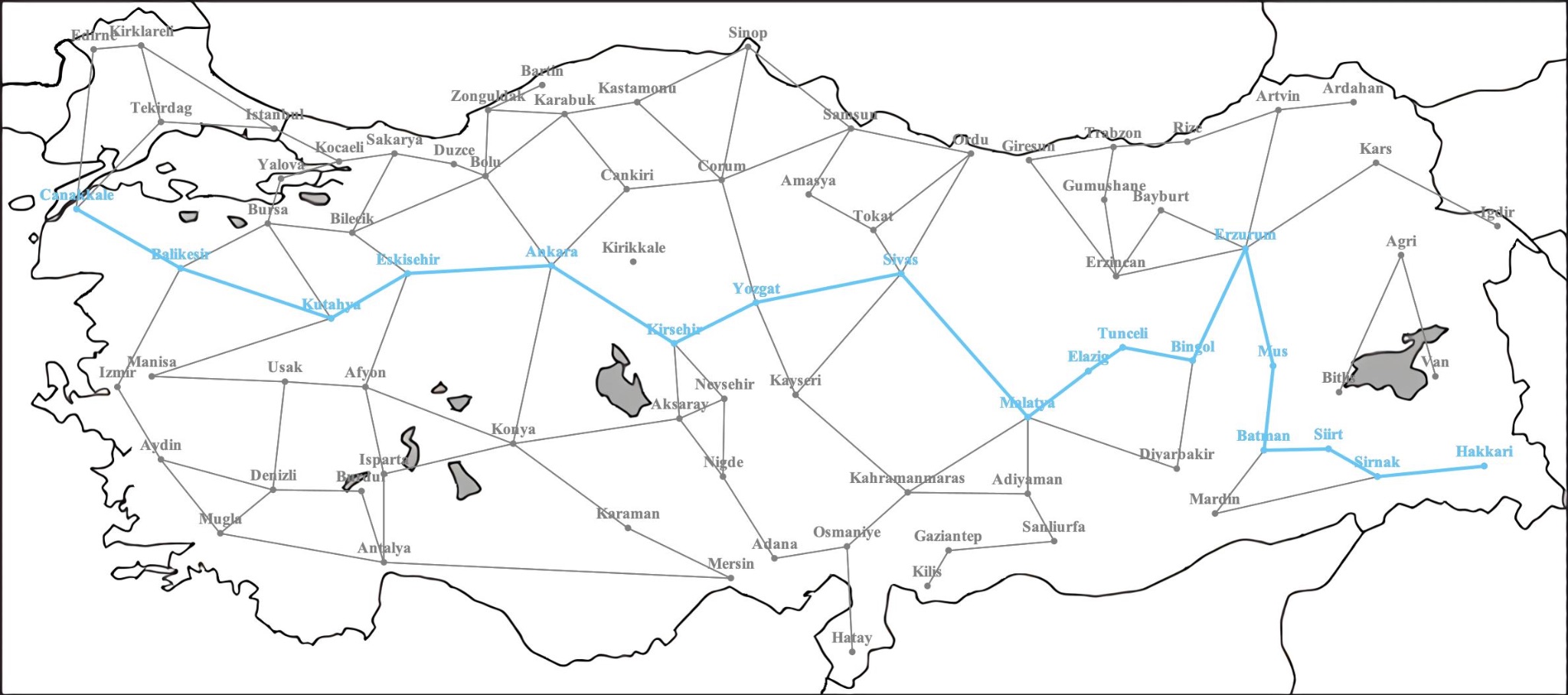
**Case 1: Edirne to Giresun**

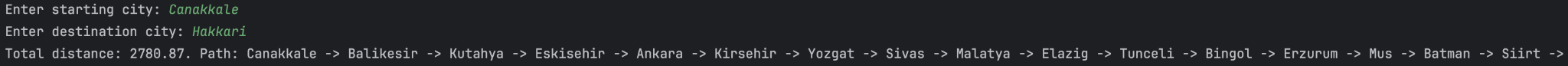






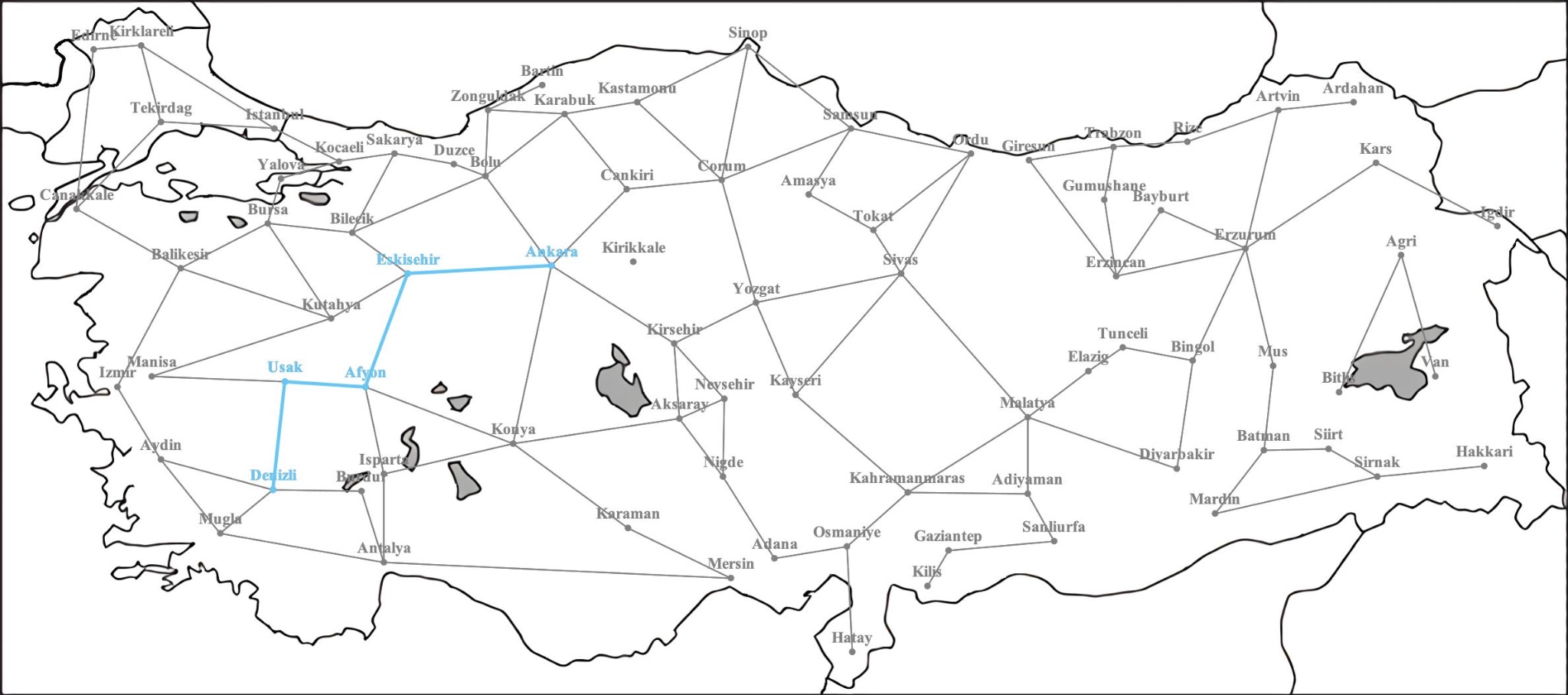
**Case 2: Canakkale to Hakkari**

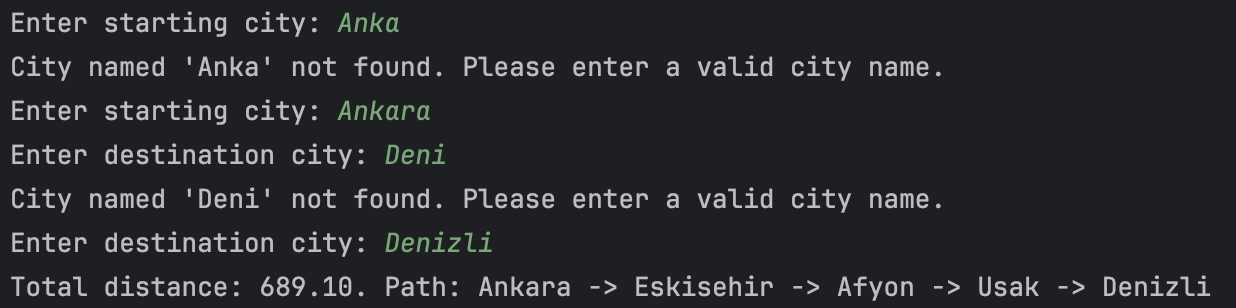




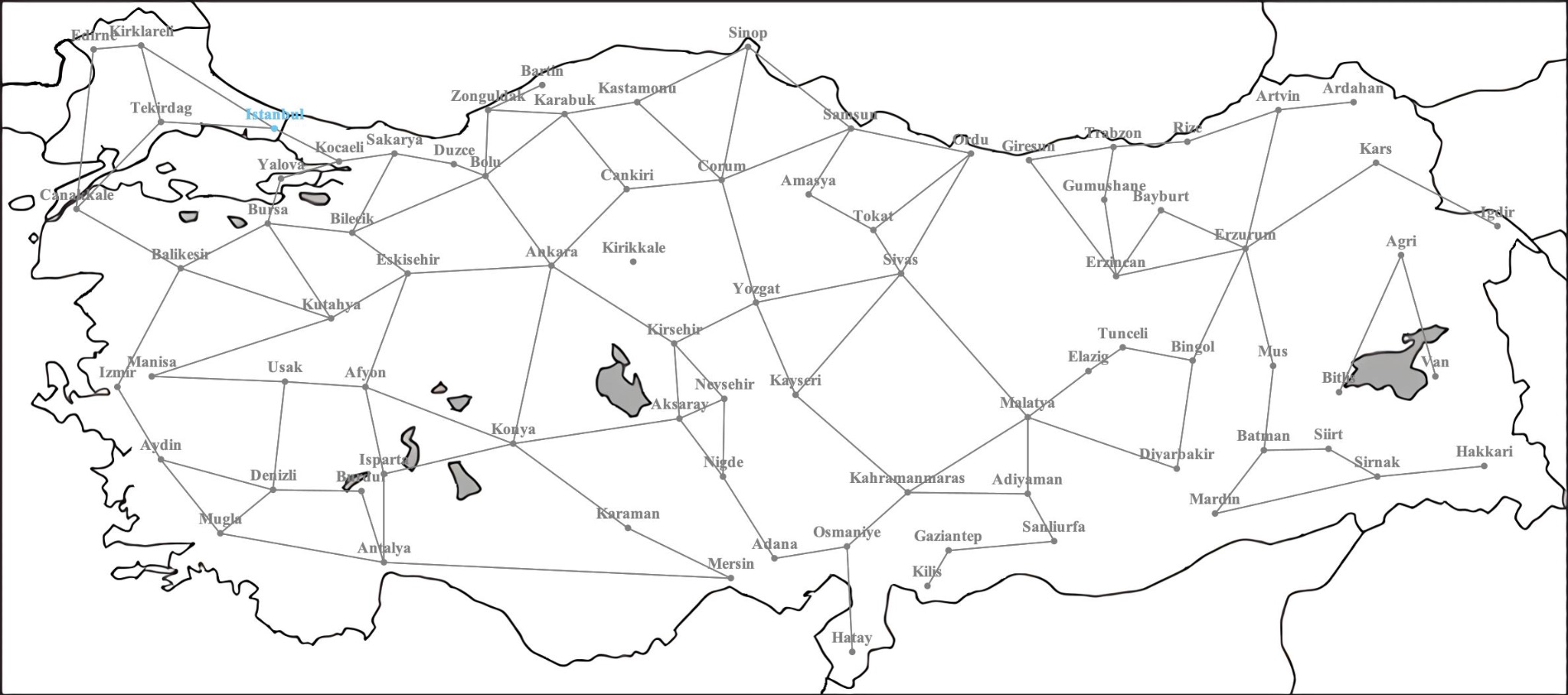


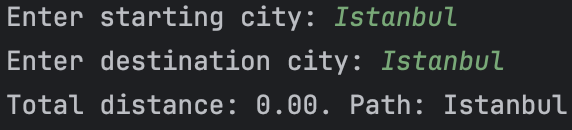
**Case 3: Ankara to Denizli (Invalid City Names)**



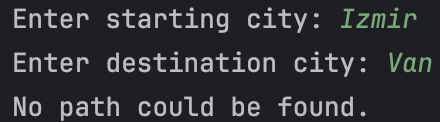


**Case 4: Istanbul (Path to the Same City)**





**Case 5: Izmir to Van (Unreachable City Pairs)**



**Dijkstra’s Algorithm:**

Dijkstra's algorithm, named after Dutch computer scientist Edsger W. Dijkstra, is a fundamental method used in graph theory and computer science to find the shortest path between two nodes in a weighted graph. It starts from a designated "source" node and systematically explores neighboring nodes, updating the shortest known distances from the source as it progresses. Dijkstra's algorithm is notable for its efficiency in finding the shortest path in graphs with non-negative edge weights. It is widely employed in various applications, including network routing protocols, transportation networks, and geographical information systems. By iteratively selecting the shortest path to unexplored nodes, Dijkstra's algorithm provides a foundational tool for solving shortest path problems in diverse domains.

**Initialization:** Assign a tentative distance value to every node, representing the shortest distance from the source node. Set source node's distance to 0 and others to infinity. Create a set of unvisited nodes.

**Iterative Process:**

Repeat until all nodes visited:

* Select unvisited node with the smallest tentative distance.
* For the current node, calculate distance to each unvisited neighbor by adding edge weight to current node's tentative distance.
* If calculated distance < previous tentative distance, update neighbor's tentative distance.

**Termination:** Mark current node as visited and remove from unvisited set. If the destination node visited or the smallest tentative distance among unvisited nodes is infinity, terminate.

**Output:** Once terminated, the shortest path from source node to every other node is known.

**Pseudocode:**

**Note:** Each distance was initialized as POSITIVE\_INFINITY when cities were created.

function dijkstra():

currentCity

nextCity

startingCity.distance = 0

startingCity.path.add(startingCity)

currentCity = startingCity

while true:

currentCity.visited = true

updateNeighbourDistances(currentCity)

nextCity = findNextCity()

if nextCity == destinationCity or nextCity == null:

break

else:

currentCity = nextCity

function updateNeighbourDistances(currentCity):

currentDistance

for city in currentCity.neighbors:

currentDistance = city.calculateDistance(currentCity) + currentCity.distance

if currentDistance < city.distance:

city.distance = currentDistance

city.path = new ArrayList<>(currentCity.path)

city.path.add(city)

function findNextCity():

minCity = null

minDist = Double.POSITIVE\_INFINITY

for city in cities:

if city.visited:

continue

if city.distance < minDist:

minDist = city.distance

minCity = city

return minCity

**References:**

[**https://en.wikipedia.org/wiki/Dijkstra's\_algorithm**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm)

[**https://www.w3schools.com/dsa/dsa\_algo\_graphs\_dijkstra.php**](https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php)

Dijkstra, E.W. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1(1), 269–271. https://doi.org/10.1007/BF01386390