import heapq

# Define the map of Romania with distances between cities

romania\_map = {

'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},

'Zerind': {'Arad': 75, 'Oradea': 71},

'Timisoara': {'Arad': 118, 'Lugoj': 111},

'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},

'Oradea': {'Zerind': 71, 'Sibiu': 151},

'Lugoj': {'Timisoara': 111, 'Mehadia': 70},

'Fagaras': {'Sibiu': 99, 'Bucharest': 211},

'Rimnicu Vilcea': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},

'Mehadia': {'Lugoj': 70, 'Drobeta': 75},

'Drobeta': {'Mehadia': 75, 'Craiova': 120},

'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},

'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},

'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},

'Giurgiu': {'Bucharest': 90}, 'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},

'Hirsova': {'Urziceni': 98, 'Eforie': 86},

'Eforie': {'Hirsova': 86},

'Vaslui': {'Urziceni': 142, 'Iasi': 92},

'Iasi': {'Vaslui': 92, 'Neamt': 87},

'Neamt': {'Iasi': 87}

}

class Node:

def \_\_init\_\_(self, city, cost, parent=None):

self.city = city

self.cost = cost

self.parent = parent

def \_\_lt\_\_(self, other):

return self.cost < other.cost

def heuristic(node, goal):

return 0 # No need for heuristic in this case

def astar\_search(graph, start, goal):

open\_list = []

closed\_set = set()

heapq.heappush(open\_list, start)

while open\_list:

current\_node = heapq.heappop(open\_list)

if current\_node.city == goal.city:

path = []

while current\_node:

path.append(current\_node.city)

current\_node = current\_node.parent

return path[::-1] # Reverse the path to get it from start to goal

closed\_set.add(current\_node.city)

for neighbor, distance in graph[current\_node.city].items():

if neighbor not in closed\_set:

new\_cost = current\_node.cost + distance

new\_node = Node(neighbor, new\_cost, current\_node)

heapq.heappush(open\_list, new\_node)

return None # No path found

start\_city = 'Arad'

goal\_city = 'Bucharest'

start\_node = Node(start\_city, 0)

goal\_node = Node(goal\_city, 0)

path = astar\_search(romania\_map, start\_node, goal\_node)

if path:

print("Path found:", path)

else:

print("No path found")