* **A\* algorithm (Find shortest path between 2 cities)**

**Code :**

import heapq

**# Define the graph of cities and their distances**

graph = {

'Mumbai': {'Pune': 150, 'Nashik': 170},

'Pune': {'Mumbai': 150, 'Sambhajinagar': 260},

'Nashik': {'Mumbai': 170, 'Sambhajinagar': 180},

'Sambhajinagar': {'Pune': 260, 'Nashik': 180}

}

**# Heuristic function to estimate distance between two cities**

heuristic = {

'Mumbai': 0,

'Pune': 120,

'Nashik': 200,

'Sambhajinagar': 300

}

def astar(start, goal):

**# Create a priority queue to store the nodes to be explored**

open\_list = [(0, start)]

**# Create a set to store the visited nodes**

closed\_list = set()

**# Create a dictionary to store the actual distance from start to each node**

g = {city: float('inf') for city in graph}

g[start] = 0

**# Create a dictionary to store the estimated total distance from start to goal via each node**

f = {city: float('inf') for city in graph}

f[start] = heuristic[start]

**# Create a dictionary to store the path taken to reach each node**

path = {start: []}

while open\_list:

**# Get the node with the lowest total estimated distance**

current\_distance, current\_city = heapq.heappop(open\_list)

**# Check if the goal is reached**

if current\_city == goal:

return g[current\_city], path[current\_city]

**# Add the current city to the closed list**

closed\_list.add(current\_city)

**# Explore the neighbors of the current city**

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

**# Calculate the actual distance from start to the neighbor**

temp\_g = g[current\_city] + distance

**# Check if the neighbor has not been visited or a shorter path is found**

if neighbor not in closed\_list and temp\_g < g[neighbor]:

**# Update the actual distance**

g[neighbor] = temp\_g

**# Update the estimated total distance**

f[neighbor] = temp\_g + heuristic[neighbor]

**# Add the neighbor to the open list**

heapq.heappush(open\_list, (f[neighbor], neighbor))

**# Update the path taken to reach the neighbor**

path[neighbor] = path[current\_city] + [(current\_city, neighbor)]

**# No path found**

return None, None

**# Print available cities**

print("Available cities:")

for city in graph.keys():

print(city)

**# Take user input for start and goal cities**

start\_city = input("Enter the start city: ")

goal\_city = input("Enter the goal city: ")

**# Find the shortest path between the user-provided cities**

shortest\_distance, shortest\_path = astar(start\_city, goal\_city)

if shortest\_distance is not None:

print(f"The shortest distance between {start\_city} and {goal\_city} is {shortest\_distance} km.")

print("The path is:")

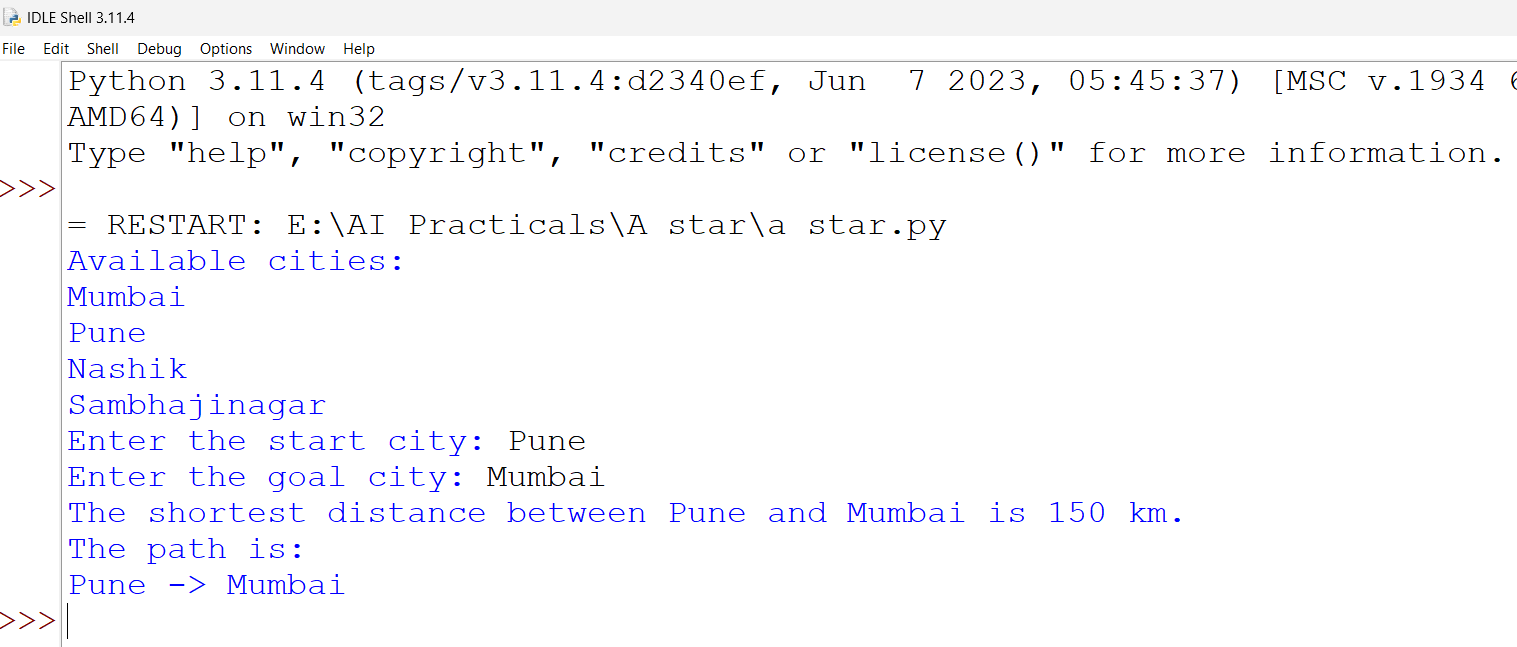
for city1, city2 in shortest\_path:

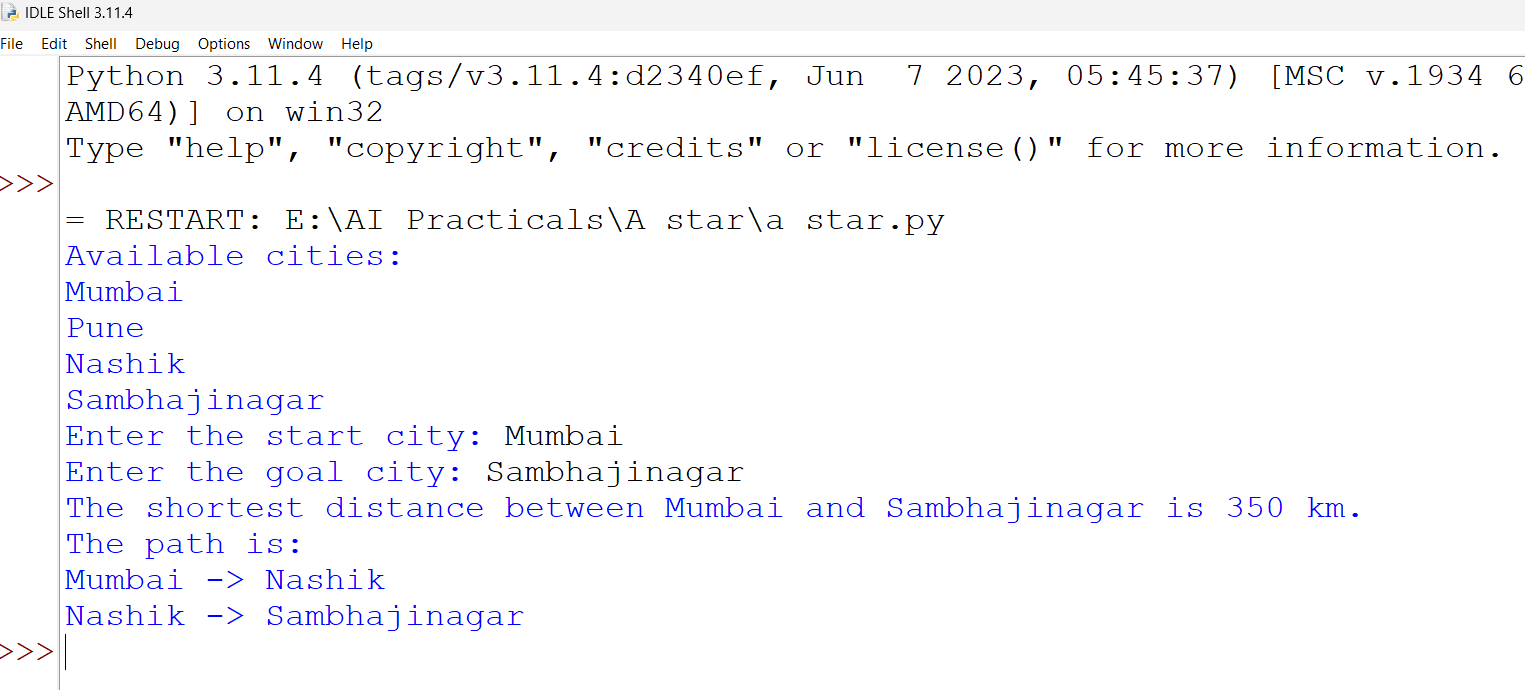
print(f"{city1} -> {city2}")

else:

print(f"No path found between {start\_city} and {goal\_city}.")

**OUTPUT**

****

****