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

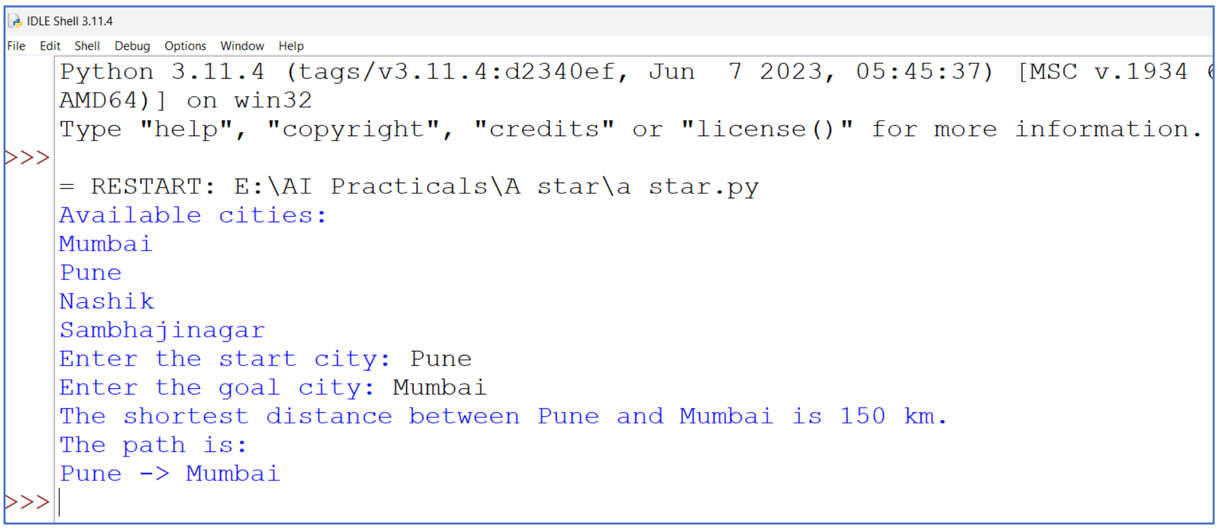
Code :

```
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
```

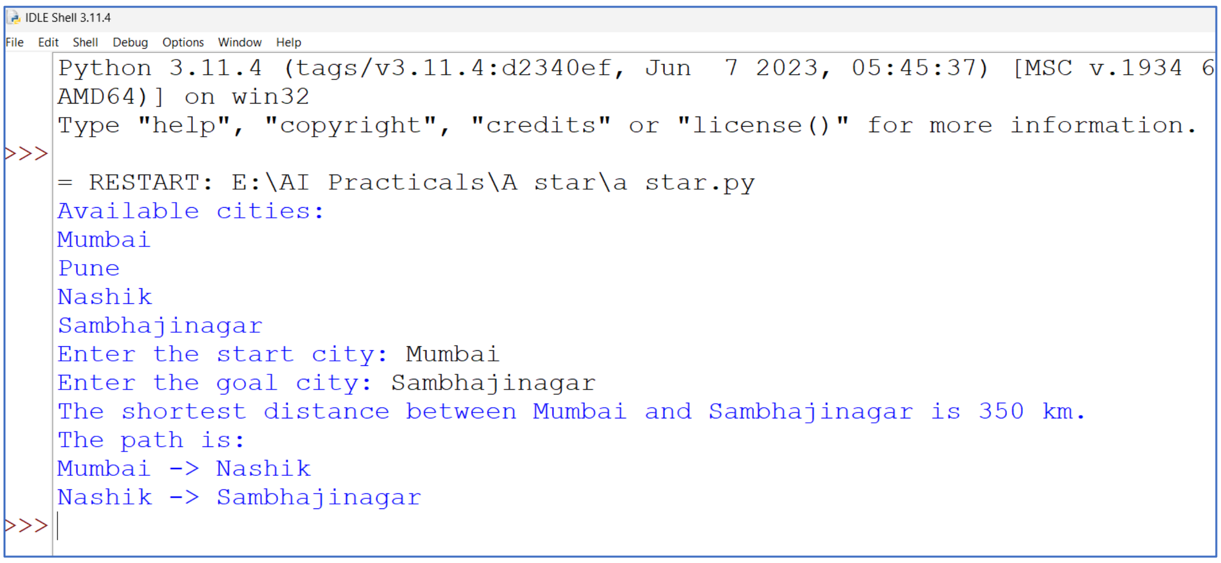
```
    path = {start: []}
    while open_list:
        # Get the node with the lowest total estimated distance
        current_distance, current_city = heapq.heappop(open_list)

        # 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



```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 6
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: E:\AI Practicals\A star\a star.py
Available cities:
Mumbai
Pune
Nashik
Sambhajinagar
Enter the start city: Pune
Enter the goal city: Mumbai
The shortest distance between Pune and Mumbai is 150 km.
The path is:
Pune -> Mumbai
>>>
```



```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 6
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: E:\AI Practicals\A star\a star.py
Available cities:
Mumbai
Pune
Nashik
Sambhajinagar
Enter the start city: Mumbai
Enter the goal city: Sambhajinagar
The shortest distance between Mumbai and Sambhajinagar is 350 km.
The path is:
Mumbai -> Nashik
Nashik -> Sambhajinagar
>>>
```