

Lab test-3

NAME: G. Bala Varshitha Reddy

HT.NO: 2403A52050

BATCH NO: AIB03

Set E11

Q1:

Prompt: A transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic data. write a python code to solve this problem

Code:

The screenshot shows a code editor window with the following Python code for a 'TransportationNetwork' class:

```
1  """
2  Transportation Route Optimization
3  Finds the shortest/fastest route between two points based on traffic congestion data.
4  Uses Dijkstra's algorithm to compute optimal paths.
5  """
6
7  import heapq
8  from collections import defaultdict
9  from typing import Dict, List, Tuple, Optional
10
11
12 class TransportationNetwork:
13     """
14         Represents a transportation network and finds optimal routes.
15     """
16
17     def __init__(self):
18         """Initialize the transportation network."""
19         self.graph = defaultdict(list) # {node: [(neighbor, distance, congestion_factor), ...]}
20         self.nodes = set()
21
22     def add_route(self, start: str, end: str, distance: float, congestion_factor: float = 1.0):
23         """
24             Add a route between two locations.
25
26             Args:
27                 start: Starting location
28                 end: Ending location
29                 distance: Distance in kilometers
30                 congestion_factor: Traffic congestion factor (1.0 = no congestion, 2.0 = heavy traffic)
31
32             # Calculate travel time: distance / average_speed * congestion_factor
33             # Assuming average speed of 60 km/h on highways
34             travel_time = (distance / 60) * congestion_factor
35
36             self.graph[start].append((end, distance, congestion_factor, travel_time))
37             self.graph[end].append((start, distance, congestion_factor, travel_time))
```

The code uses Dijkstra's algorithm to find the shortest or fastest path between nodes in a graph where edges have weights based on distance and congestion factors.

A screenshot of Microsoft Visual Studio Code showing a Python file named `task_1.py`. The code implements a `TransportationNetwork` class with methods to add routes and find the fastest route using Dijkstra's algorithm. A task pane on the right provides context about finding the shortest/fastest route between two points based on traffic congestion data.

```
lab_exam-3 > task_1.py ...
12  class TransportationNetwork:
13      def add_route(self, start: str, end: str, distance: float, congestion_factor: float = 1.0):
14          self.graph[end].append((start, distance, congestion_factor, travel_time))
15          self.nodes.add(start)
16          self.nodes.add(end)
17
18      def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
19          """
20              Find the fastest route from start to end using Dijkstra's algorithm.
21
22              Args:
23                  start: Starting location
24                  end: Destination location
25
26              Returns:
27                  Tuple of (route_path, total_time, total_distance) or (None, None, None) if no path exists
28
29          if start not in self.nodes or end not in self.nodes:
30              print(f"Error: Start '{start}' or End '{end}' not in network")
31              return None, None, None
32
33          # Initialize distances and previous nodes
34          distances = {node: float('inf') for node in self.nodes}
35          distances[start] = 0
36          previous = {node: None for node in self.nodes}
37          visited = set()
38
39          # Priority queue: (time, node)
40          pq = [(0, start)]
41
42          while pq:
43              current_time, current_node = heapq.heappop(pq)
44
45              if current_node in visited:
46                  continue
47
48              visited.add(current_node)
49
50              # If we reached the destination, we can stop
51              if current_node == end:
52                  break
53
54              # Check all neighbors
55              for neighbor, distance, _, _ in self.graph[current_node]:
56                  if neighbor in visited:
57                      continue
58
59                  new_time = current_time + distance
60
61                  if new_time < distances[neighbor]:
62                      distances[neighbor] = new_time
63                      previous[neighbor] = current_node
64                      heapq.heappush(pq, (new_time, neighbor))
65
66
67          # Reconstruct the path
68          if distances[end] == float('inf'):
69              print(f"No route found from '{start}' to '{end}'")
70              return None, None, None
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
```

A screenshot of Microsoft Visual Studio Code showing the same Python file `task_1.py`. In this version, the code has been annotated with several highlights in yellow and red, likely indicating areas of interest or error detection by the AI tool. The annotations include highlights around the `TransportationNetwork` class definition, the `add_route` method, the `find_fastest_route` method, and various sections of the Dijkstra's algorithm implementation.

```
lab_exam-3 > task_1.py ...
12  class TransportationNetwork:
13      def add_route(self, start: str, end: str, distance: float, congestion_factor: float = 1.0):
14          self.graph[end].append((start, distance, congestion_factor, travel_time))
15          self.nodes.add(start)
16          self.nodes.add(end)
17
18      def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
19          """
20              Find the fastest route from start to end using Dijkstra's algorithm.
21
22              Args:
23                  start: Starting location
24                  end: Destination location
25
26              Returns:
27                  Tuple of (route_path, total_time, total_distance) or (None, None, None) if no path exists
28
29          if start not in self.nodes or end not in self.nodes:
30              print(f"Error: Start '{start}' or End '{end}' not in network")
31              return None, None, None
32
33          # Initialize distances and previous nodes
34          distances = {node: float('inf') for node in self.nodes}
35          distances[start] = 0
36          previous = {node: None for node in self.nodes}
37          visited = set()
38
39          # Priority queue: (time, node)
40          pq = [(0, start)]
41
42          while pq:
43              current_time, current_node = heapq.heappop(pq)
44
45              if current_node in visited:
46                  continue
47
48              visited.add(current_node)
49
50              # If we reached the destination, we can stop
51              if current_node == end:
52                  break
53
54              # Check all neighbors
55              for neighbor, distance, _, _ in self.graph[current_node]:
56                  if neighbor in visited:
57                      continue
58
59                  new_time = current_time + distance
60
61                  if new_time < distances[neighbor]:
62                      distances[neighbor] = new_time
63                      previous[neighbor] = current_node
64                      heapq.heappush(pq, (new_time, neighbor))
65
66
67          # Reconstruct the path
68          if distances[end] == float('inf'):
69              print(f"No route found from '{start}' to '{end}'")
70              return None, None, None
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
```

File Edit Selection View Go Run Terminal Help ← → Q AI

task_1.py

```
lab_exam-3 > task_1.py > ...
12 class TransportationNetwork:
13     def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
14         """
15             break
16             current = previous[current]
17
18             path.reverse()
19             return path, distances[end], total_distance
20
21         def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
22             """
23                 Find the shortest distance route from start to end.
24
25                 Args:
26                     start: Starting location
27                     end: Destination location
28
29                 Returns:
30                     Tuple of (route_path, total_distance, total_time) or (None, None, None) if no path exists
31
32             if start not in self.nodes or end not in self.nodes:
33                 print(f"Error: Start '{start}' or End '{end}' not in network")
34                 return None, None, None
35
36             # Initialize distances and previous nodes
37             distances = {node: float('inf') for node in self.nodes}
38             distances[start] = 0
39             previous = {node: None for node in self.nodes}
40             visited = set()
41
42             # Priority queue: (distance, node)
43             pq = [(0, start)]
44
45             while pq:
46                 current_dist, current_node = heapq.heappop(pq)
47
48                 if current_node in visited:
49                     continue
50
51                 visited.add(current_node)
52
53                 if current_node == end:
54                     break
55
56                 for neighbor, distance, congestion, _ in self.graph[current_node]:
57                     if neighbor in visited:
58                         continue
59
60                     new_distance = current_dist + distance
61
62                     if new_distance < distances[neighbor]:
63                         distances[neighbor] = new_distance
64                         previous[neighbor] = current_node
65                         heapq.heappush(pq, (new_distance, neighbor))
66
67             # Reconstruct the path
68             if distances[end] == float('inf'):
69                 print(f"No route found from '{start}' to '{end}'")
70                 return None, None, None
71
72             path = []
73             current = end
74             total_time = 0
75
76             while current is not None:
77                 path.append(current)
78                 if previous[current] is not None:
79                     for neighbor, distance, congestion, travel_time in self.graph[current]:
80                         if neighbor == previous[current]:
81                             total_time += travel_time
82                             break
83                     current = previous[current]
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

Enable Claude Haiku 4.5 for all clients
Enable access to the latest Claude Haiku 4.5 model from Anthropic. Learn more about how GitHub Copilot serves Claude Haiku 4.5.

Varshitha-bit select

File Edit Selection View Go Run Terminal Help ← → Q AI

task_1.py

```
lab_exam-3 > task_1.py > ...
12 class TransportationNetwork:
13     def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
14         """
15             continue
16             visited.add(current_node)
17
18             if current_node == end:
19                 break
20
21             for neighbor, distance, congestion, _ in self.graph[current_node]:
22                 if neighbor in visited:
23                     continue
24
25                 new_distance = current_dist + distance
26
27                 if new_distance < distances[neighbor]:
28                     distances[neighbor] = new_distance
29                     previous[neighbor] = current_node
30                     heapq.heappush(pq, (new_distance, neighbor))
31
32             # Reconstruct the path
33             if distances[end] == float('inf'):
34                 print(f"No route found from '{start}' to '{end}'")
35                 return None, None, None
36
37             path = []
38             current = end
39             total_time = 0
40
41             while current is not None:
42                 path.append(current)
43                 if previous[current] is not None:
44                     for neighbor, distance, congestion, travel_time in self.graph[current]:
45                         if neighbor == previous[current]:
46                             total_time += travel_time
47                             break
48                     current = previous[current]
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

Enable Claude Haiku 4.5 for all clients
Enable access to the latest Claude Haiku 4.5 model from Anthropic. Learn more about how GitHub Copilot serves Claude Haiku 4.5.

Varshitha-bit select

```
lab_exam-3 > task_1.py > ...
12     class TransportationNetwork:
13         def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
14             current = previous[current]
15
16             path.reverse()
17             return path, distances[end], total_time
18
19         def display_network(self):
20             """Display all routes in the network."""
21             print("\n" + "*" * 60)
22             print("TRANSPORTATION NETWORK ROUTES")
23             print("*" * 60)
24             for node in sorted(self.graph.keys()):
25                 print(f"\nFrom {node}:")
26                 for neighbor, distance, congestion, travel_time in self.graph[node]:
27                     print(f"    {neighbor}: {distance} km, Congestion: {congestion}x, Time: {travel_time:.2f} hours")
28
29
30     def main():
31         """Main function demonstrating route optimization."""
32
33         # Create transportation network
34         network = TransportationNetwork()
35
36         # Add routes with distance and congestion factors
37         # Format: (start, end, distance_km, congestion_factor)
38         routes = [
39             ("A", "B", 10, 1.0),      # Clear road
40             ("A", "C", 15, 1.5),    # Moderate traffic
41             ("B", "D", 12, 2.0),    # Heavy traffic
42             ("C", "D", 8, 1.2),     # Light traffic
43             ("B", "E", 20, 1.0),    # Clear road
44             ("C", "E", 18, 1.8),    # Congested
45             ("D", "E", 5, 1.5),     # Moderate traffic
46             ("A", "D", 25, 2.5),    # Very heavy traffic
47         ]
48
49
50
51
52
53
54
55
56
57
58
59
59
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

task_1.py

Enable Claude Haiku 4.5 for all clients

Enable access to the latest Claude Haiku 4.5 model from Anthropic.

Learn more about how GitHub Copilot serves Claude Haiku 4.5.

Varsitha-bit selected

task_1.py +

Add context (#), enter

A J V D v

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "lab_exam-3" containing "task_1.py".
- Search Bar:** Contains the text "AI".
- Code Editor:** Displays Python code for route planning. The code includes functions for adding routes to a network, displaying the network, finding the fastest route, and finding the shortest route between locations A and E.
- Output Panel:** Shows the results of running the code, including the routes, travel times, and total distances for both the fastest and shortest routes.
- Bottom Status Bar:** Shows "In 255 Col 1 Spaces:4 UFT-8 Python 3.11.9 (Microsoft Store)".

```

lab_exam-3 > task_1.py > ...
190 def main():
191     start_location = "A"
192     end_location = "E"
193
194     fastest_path, fastest_time, fastest_distance = network.find_fastest_route(start_location, end_location)
195
196     if fastest_path:
197         print(f"\nFastest Route: (' + '.join(fastest_path))")
198         print(f"Travel Time: {fastest_time:.2f} hours ({fastest_time * 60:.0f} minutes)")
199         print(f"Total Distance: {fastest_distance:.2f} km")
200
201     # Find shortest route
202     print("\n" + "="*60)
203     print("FINDING SHORTEST ROUTE FROM A TO E")
204     print("="*60)
205
206     shortest_path, shortest_distance, shortest_time = network.find_shortest_route(start_location, end_location)
207
208     if shortest_path:
209         print(f"\nShortest Route: (' + '.join(shortest_path))")
210         print(f"Total Distance: {shortest_distance:.2f} km")
211         print(f"Travel Time: {shortest_time:.2f} hours ({shortest_time * 60:.0f} minutes)")
212
213     # Compare routes
214     print("\n" + "="*60)
215     print("ROUTE COMPARISON")
216     print("="*60)
217     if fastest_path and shortest_path:
218         time_saved = (shortest_time - fastest_time) * 60
219         dist_diff = shortest_distance - fastest_distance
220         print(f"\nFastest Route saves {abs(time_saved):.0f} minutes compared to shortest route")
221         print(f"Fastest route covers {dist_diff:.2f} km ('more' if dist_diff > 0 else 'less') distance")
222
223     if __name__ == "__main__":
224         main()

```

Output:

```

2 Transportation Route Optimization
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + | ... | x
PS C:\Users\varsh\Desktop\AI> & 'c:\Users\varsh\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\varsh\vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '5230
● 1' '--' 'C:\Users\varsh\Desktop\AI\lab_exam-3\task_1.py'

=====
TRANSPORTATION NETWORK ROUTES
=====

From A:
→ B: 10 km, Congestion: 1.0x, Time: 0.17 hours
→ C: 15 km, Congestion: 1.5x, Time: 0.38 hours
→ D: 25 km, Congestion: 2.5x, Time: 1.04 hours

From B:
→ A: 10 km, Congestion: 1.0x, Time: 0.17 hours
→ D: 12 km, Congestion: 2.0x, Time: 0.40 hours
→ E: 20 km, Congestion: 1.0x, Time: 0.33 hours

From C:
→ A: 15 km, Congestion: 1.5x, Time: 0.38 hours
→ D: 8 km, Congestion: 1.2x, Time: 0.16 hours
→ E: 18 km, Congestion: 1.8x, Time: 0.54 hours

From D:
→ B: 12 km, Congestion: 2.0x, Time: 0.40 hours
→ C: 8 km, Congestion: 1.2x, Time: 0.16 hours
→ E: 5 km, Congestion: 1.5x, Time: 0.12 hours
→ A: 25 km, Congestion: 2.5x, Time: 1.04 hours

From E:
→ B: 20 km, Congestion: 1.0x, Time: 0.33 hours
→ C: 18 km, Congestion: 1.8x, Time: 0.54 hours
→ D: 5 km, Congestion: 1.5x, Time: 0.12 hours
=====
```

```
→ C: 18 km, Congestion: 1.8x, Time: 0.54 hours  
→ D: 5 km, Congestion: 1.5x, Time: 0.12 hours
```

```
=====  
FINDING FASTEST ROUTE FROM A TO E  
=====
```

```
Fastest Route: A → B → E  
Travel Time: 0.50 hours (30 minutes)  
Total Distance: 30.00 km
```

```
=====  
FINDING SHORTEST ROUTE FROM A TO E  
=====
```

```
Shortest Route: A → B → D → E  
Total Distance: 27.00 km  
Travel Time: 0.69 hours (42 minutes)
```

```
=====  
ROUTE COMPARISON  
=====
```

```
Fastest Route saves 12 minutes compared to shortest route  
Fastest route covers -3.00 km less distance
```

```
○ PS C:\Users\varsh\Desktop\AI>
```

Observation:

The main idea of the code is to compare two possible routes between locations, the route with the shortest distance and the route with the fastest travel time, considering traffic. AI helps by using data about road traffic to pick the fastest route, instead of just the shortest one. The algorithm checks time and distance for each path and decides which is better for quick transportation. This helps companies avoid delays from traffic and makes deliveries smarter and more efficient.

Q2:

Prompt:

A transportation company wants to find the best route between two places using traffic data. Design a backend API that takes the start and end points, and returns the best route

Code:

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a project structure with files like `task_1.py`, `task_2.py`, `app.py`, and various PDF documents.
- Code Editor:** Displays the content of `task_2.py`. The code uses the Flask framework to handle a POST request to `/api/best-route`. It defines a simulated traffic and route data dictionary and uses it to find the route with the least travel time.
- Terminal:** Shows a command prompt at the bottom with the path `C:\Users\varsh\Desktop\AI>`.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS, along with a Python Debug session and a PowerShell session.

```
1  from flask import Flask, request, jsonify
2
3  app = Flask(__name__)
4
5  # Simulated traffic and route data
6  routes = {
7      "A-B-D": {"distance": 22, "time": 25},    # Route 1
8      "A-C-D": {"distance": 25, "time": 20}     # Route 2 (faster, but longer)
9  }
10
11 @app.route('/api/best-route', methods=['POST'])
12 def best_route():
13     # Example input: { "from": "A", "to": "D" }
14     data = request.get_json()
15     if not data or data["from"] != "A" or data["to"] != "D":
16         return jsonify({"error": "Invalid input"}), 400
17
18     # AI logic: pick route with least time
19     best = min(routes.items(), key=lambda x: x[1]["time"])
20     return jsonify({
21         "route": best[0],
22         "distance_km": best[1]["distance"],
23         "travel_time_min": best[1]["time"]
24     })
25
26 if __name__ == '__main__':
27     app.run(debug=True)
```

Output:

The screenshot shows the Postman interface. In the left sidebar, there is a collection named "My Collection" with a "POST Post data" endpoint selected. The main area shows a POST request to "http://127.0.0.1:5000/api/best-route". The "Body" tab is active, displaying the following JSON input:

```
1 {
2   "from": "A",
3   "to": "D"
4 }
```

Below the request, the response is shown with a status of "200 OK". The response body is also JSON:

```
1 {
2   "distance_km": 25,
3   "route": "A-C-D",
4   "travel_time_min": 20
5 }
```

Observation:

The backend API helps the company find the best route between two points by using real-time traffic data and AI-assisted decisions. To test the API, we use the URL shown in the terminal after running the Flask app. We will send the required input through Postman or any client, and the API replies with the best route, distance, and travel time. This makes route planning faster and smarter, improving company efficiency.