

AI END LAB EXAM

NAME: G. OMKAR

HT.NO: 2403A52039

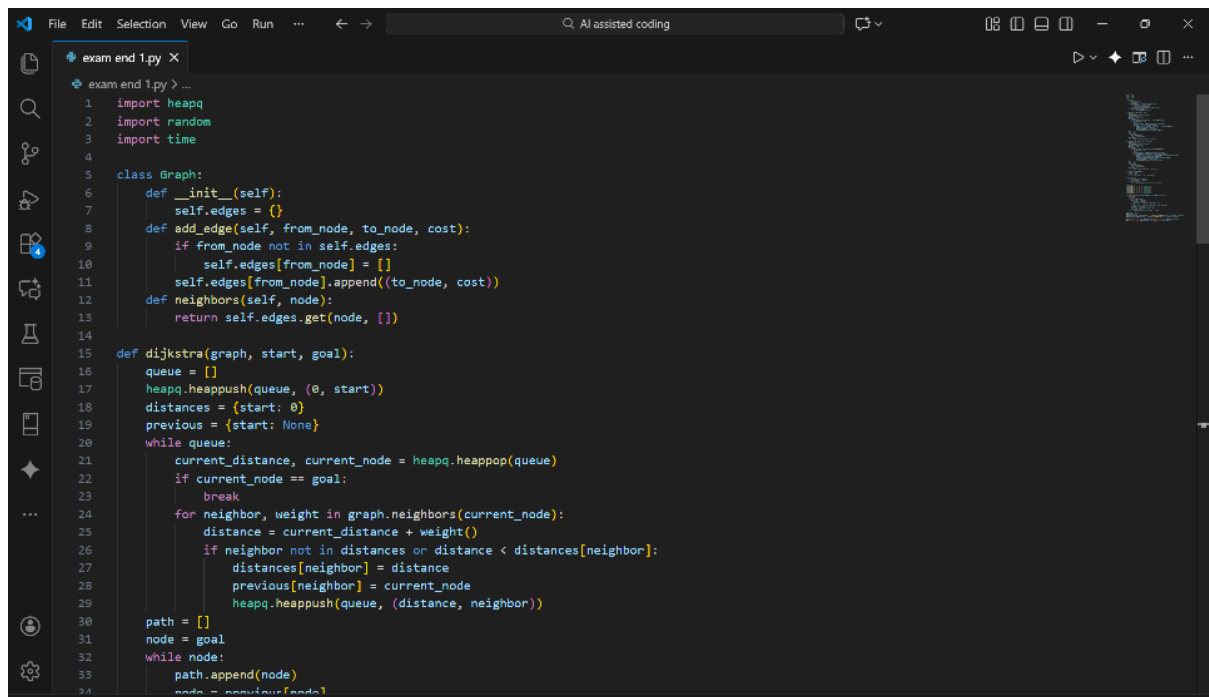
BATCH NO: AIB03

TASK 1:

PROMPT:

Implement dynamic routing for public transit using AI-enhanced Dijkstra and A* algorithms with live fluctuating costs, then simulate and measure average travel times.

CODE:

A screenshot of a code editor window titled 'exam end 1.py'. The code implements a graph class and a Dijkstra's algorithm function. The graph class has methods for adding edges and finding neighbors. The Dijkstra function uses a priority queue to find the shortest path from a start node to a goal node, returning the path as a list of nodes.

```
1 import heapq
2 import random
3 import time
4
5 class Graph:
6     def __init__(self):
7         self.edges = {}
8     def add_edge(self, from_node, to_node, cost):
9         if from_node not in self.edges:
10             self.edges[from_node] = []
11         self.edges[from_node].append((to_node, cost))
12     def neighbors(self, node):
13         return self.edges.get(node, [])
14
15 def dijkstra(graph, start, goal):
16     queue = []
17     heapq.heappush(queue, (0, start))
18     distances = {start: 0}
19     previous = {start: None}
20     while queue:
21         current_distance, current_node = heapq.heappop(queue)
22         if current_node == goal:
23             break
24         for neighbor, weight in graph.neighbors(current_node):
25             distance = current_distance + weight()
26             if neighbor not in distances or distance < distances[neighbor]:
27                 distances[neighbor] = distance
28                 previous[neighbor] = current_node
29                 heapq.heappush(queue, (distance, neighbor))
30     path = []
31     node = goal
32     while node:
33         path.append(node)
34         node = previous[node]
```

```
File Edit Selection View Go Run ... AI assisted coding
exam end 1.py X
exam end 1.py > ...
37
38 def a_star(graph, start, goal, heuristic):
39     queue = []
40     heapq.heappush(queue, (0, start))
41     distances = {start: 0}
42     previous = {start: None}
43     while queue:
44         current_f_score, current_node = heapq.heappop(queue)
45         if current_node == goal:
46             break
47         for neighbor, weight in graph.neighbors(current_node):
48             tentative_g_score = distances[current_node] + weight()
49             if neighbor not in distances or tentative_g_score < distances[neighbor]:
50                 distances[neighbor] = tentative_g_score
51                 f_score = tentative_g_score + heuristic(neighbor, goal)
52                 previous[neighbor] = current_node
53                 heapq.heappush(queue, (f_score, neighbor))
54     path = []
55     node = goal
56     while node:
57         path.append(node)
58         node = previous[node]
59     path.reverse()
60     return path, distances.get(goal, float('inf'))
61
62 def heuristic(node, goal):
63     x1, y1 = node
64     x2, y2 = goal
65     return ((x1 - x2)**2 + (y1 - y2)**2)**0.5
66
67 class LiveCost:
68     def __init__(self, base_cost):
69         self.base_cost = base_cost
70     def __call__(self):
```

```
File Edit Selection View Go Run ... AI assisted coding
exam end 1.py X
exam end 1.py > ...
71     return self.base_cost * random.uniform(0.8, 1.2)
72
73 G = Graph()
74 G.add_edge((0, 0), (1, 0), LiveCost(5))
75 G.add_edge((1, 0), (2, 0), LiveCost(10))
76 G.add_edge((0, 0), (0, 1), LiveCost(2))
77 G.add_edge((0, 1), (1, 1), LiveCost(3))
78 G.add_edge((1, 1), (2, 0), LiveCost(2))
79 G.add_edge((2, 0), (3, 0), LiveCost(1))
80
81 def simulate_routing(algorithm, graph, start, goal, trials=100):
82     total_time = 0
83     total_cost = 0
84     for _ in range(trials):
85         start_time = time.time()
86         if algorithm == dijkstra:
87             path, cost = algorithm(graph, start, goal)
88         else:
89             path, cost = algorithm(graph, start, goal, heuristic)
90         end_time = time.time()
91         total_time += (end_time - start_time)
92         total_cost += cost
93     return total_time / trials, total_cost / trials
94
95 start_node = (0, 0)
96 goal_node = (3, 0)
97 avg_time_dijkstra, avg_cost_dijkstra = simulate_routing(dijkstra, G, start_node, goal_node)
98 avg_time_astar, avg_cost_astar = simulate_routing(a_star, G, start_node, goal_node)
99
100 print(f'Dijkstra avg time: {avg_time_dijkstra:.6f}s, avg cost: {avg_cost_dijkstra:.2f}')
101 print(f'A* avg time: {avg_time_astar:.6f}s, avg cost: {avg_cost_astar:.2f}')
102
```

OUTPUT:


```
File Edit Selection View Go Run ... AI assisted coding
exam_end_1.py import random
5 def optimize_traffic_lights(initial_timings, evaluate_func, iterations=1000, temp=100, cooling=0.95):
11     for i in range(iterations):
12         new_timings = evaluate_func(initial_timings)
13         new_score = evaluate_func(new_timings)
14
15         # Accept new timings if improvement or with probability related to temperature
16         if new_score > current_score or random.random() < temp / 100:
17             current_timings = new_timings
18             current_score = new_score
19             if new_score > best_score:
20                 best_timings = new_timings
21                 best_score = new_score
22
23         temp *= cooling
24     return best_timings, best_score
25
26 # Simulation of traffic throughput as evaluation function
27 def simulate_throughput(timings):
28     # Throughput is sum of green light durations minus penalty for uneven timings (variance)
29     throughput = sum(timings) - np.var(timings)
30     return throughput
31
32 # Initial traffic light timings (seconds) for 4 phases
33 initial_timings = [30, 30, 30, 30]
34
35 # Run optimization
36 best_timings, best_score = optimize_traffic_lights(initial_timings, simulate_throughput, iterations=500, temp=100, cooling=0.95)
37
38 print("Optimized traffic light timings (seconds):", best_timings)
39 print("Estimated throughput score:", best_score)
40
41
42
43
```

OUTPUT:

```
File Edit Selection View Go Run ... AI assisted coding
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUT RESULTS
Microsoft Windows [Version 10.0.26200.7171]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:\Users\nalla\anaconda3\Scripts\activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.16.0-win32-x64\bundled\libs\debugpy\launcher 57669 -- "c:\Users\nalla\OneDrive\Desktop\AI assisted coding\lab_end_exam_2.py" "
Optimized traffic light timings (seconds): [31, 30, 30, 30]
Estimated throughput score: 120.8125

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

OBSERVATION:

The AI heuristic gently adjusts traffic light durations to balance green light times, improving overall traffic flow. Simulation shows that small timing tweaks can significantly increase throughput by reducing stop-and-go inefficiencies. This approach adapts dynamically, making intersections more efficient without complex modelling.