DATE:25/07/2025

EXP:03

Autonomous Vehicle Route Planning Using A* Algorithm

AIM:

Problem statement

Develop a simplified smart route planner for an autonomous vehicle navigating a city. The city map is modeled as a weighted undirected graph, where:

- Nodes represent intersections or landmarks
- Edges represent roads connecting intersections
- Weights represent travel time or traffic congestion on each road

Implement the A* search algorithm to compute the optimal (least time-cost) route from a source node to a destination node, considering road weights and using a straight-line heuristic between points.

Objective

- Model the city as a weighted graph with node coordinates.
- Apply A* to find the least-cost path between two nodes.
- Use Euclidean distance as the heuristic.
- Display the computed route, total travel cost, and number of nodes explored.

ALGORITHM:

Algorithm: A* Search

Input:

- Start node start
- Goal node goal
- A heuristic function h(n) that estimates cost from node n to the goal
- Step-cost function g(n) (actual cost so far from start to node n)

Output:

- Optimal path from start to goal (if one exists)

Procedure

1. Initialize an open list (priority queue) and add start with:
   - g(start) = 0
   - f(start) = g(start) + h(start)
2. Initialize a closed set (visited nodes) as empty.
3. While open list is not empty:
   a. Remove the node n from open list with the lowest f(n).
   b. If n is the goal node, return the path from start to goal.
   c. Add n to the closed set.
   d. For each successor s of n:
   - Compute g(s) = g(n) + cost(n, s)
   - Compute f(s) = g(s) + h(s)
   - If s is already in closed set with a lower f(s), skip it.
   - If s is not in open list (or has better f(s) than before), insert/update it in open list.
4. If open list becomes empty and goal not found → Failure (no solution).

CODE:

```
#A*

import heapq

import math

# Coordinates for intersections

coordinates = {

    'A': (0, 0),

    'B': (2, 2),
```

```
    'C': (2, 0),
    'D': (4, 2),
    'E': (5, 0),
    'F': (6, 2)
}
# Weighted undirected graph (travel time or congestion)
graph = {
    'A': [('B', 3), ('C', 1)],
    'B': [('A', 3), ('D', 3)],
    'C': [('A', 1), ('D', 4), ('E', 2)],
    'D': [('B', 3), ('C', 4), ('F', 1)],
    'E': [('C', 2), ('F', 5)],
    'F': [('D', 1), ('E', 5)]
}
start = 'A'
goal = 'F'
def heuristic(node1, node2):
    x1, y1 = coordinates[node1]
    x2, y2 = coordinates[node2]
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def a_star_verbose(graph, start, goal):
    open_set = []
    heapq.heappush(open_set, (heuristic(start, goal), 0, start, [start]))
```

```
visited = set()
nodes_expanded = 0
step_count = 1
while open_set:
    f, g, current, path = heapq.heappop(open_set)
    if current in visited:
        continue
    visited.add(current)
    nodes_expanded += 1
    # Step-by-step debug output
    print(f"\n Step {step_count}: Visiting Node '{current}'")
    print(f"  g(n) = {g:.2f} (Cost from start)")
    h = heuristic(current, goal)
    print(f"  h(n) = {h:.2f} (Heuristic to goal)")
    print(f"  f(n) = g + h = {g:.2f} + {h:.2f} = {f:.2f}")
    print(f"  Path so far: {' → '.join(path)}")
    # Show current queue status
    print("  📥 Queue Contents:")
    for est, cost, node, p in open_set:
        print(f"    {node}: f={est:.2f}, g={cost}, path={' → '.join(p)}")
    step_count += 1

    if current == goal:
```

```
            print(f"\n Goal '{goal}' reached!")
            return path, g, nodes_expanded
        for neighbor, weight in graph[current]:
            if neighbor not in visited:
                g_new = g + weight
                f_new = g_new + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_new, g_new, neighbor, path + [neighbor]))
    return None, float('inf'), nodes_expanded
# Run the A* search with explanation
final_path, total_cost, explored = a_star_verbose(graph, start, goal)
# Final result
print("\n======================")
if final_path:
    print(" Path Found:", " → ".join(final_path))
    print(" Total Travel Time:", total_cost)
    print(" Nodes Expanded:", explored)
else:
    print(" No path found.")
```

OUTPUT:

```
Step 1: Visiting Node 'A'
 g(n) = 0.00 (Cost from start)
 h(n) = 6.32 (Heuristic to goal)
 f(n) = g + h = 0.00 + 6.32 = 6.32
 Path so far: A
 📥 Queue Contents:

Step 2: Visiting Node 'C'
 g(n) = 1.00 (Cost from start)
 h(n) = 4.47 (Heuristic to goal)
 f(n) = g + h = 1.00 + 4.47 = 5.47
 Path so far: A → C
 📥 Queue Contents:
   B: f=7.00, g=3, path=A → B

Step 3: Visiting Node 'E'
 g(n) = 3.00 (Cost from start)
 h(n) = 2.24 (Heuristic to goal)
 f(n) = g + h = 3.00 + 2.24 = 5.24
 Path so far: A → C → E
 📥 Queue Contents:
   B: f=7.00, g=3, path=A → B
   D: f=7.00, g=5, path=A → C → D

Step 4: Visiting Node 'B'
 g(n) = 3.00 (Cost from start)
 h(n) = 4.00 (Heuristic to goal)
 f(n) = g + h = 3.00 + 4.00 = 7.00
 Path so far: A → B
 📥 Queue Contents:
   D: f=7.00, g=5, path=A → C → D
   F: f=8.00, g=8, path=A → C → E → F
```

```
Step 5: Visiting Node 'D'
 g(n) = 5.00 (Cost from start)
 h(n) = 2.00 (Heuristic to goal)
 f(n) = g + h = 5.00 + 2.00 = 7.00
 Path so far: A → C → D
 📥 Queue Contents:
   D: f=8.00, g=6, path=A → B → D
   F: f=8.00, g=8, path=A → C → E → F

Step 6: Visiting Node 'F'
 g(n) = 6.00 (Cost from start)
 h(n) = 0.00 (Heuristic to goal)
 f(n) = g + h = 6.00 + 0.00 = 6.00
 Path so far: A → C → D → F
 📥 Queue Contents:
   D: f=8.00, g=6, path=A → B → D
   F: f=8.00, g=8, path=A → C → E → F

Goal 'F' reached!

======================
Path Found: A → C → D → F
Total Travel Time: 6
Nodes Expanded: 6
```

RESULT:

The programs have been completed and the outputs have been verified.