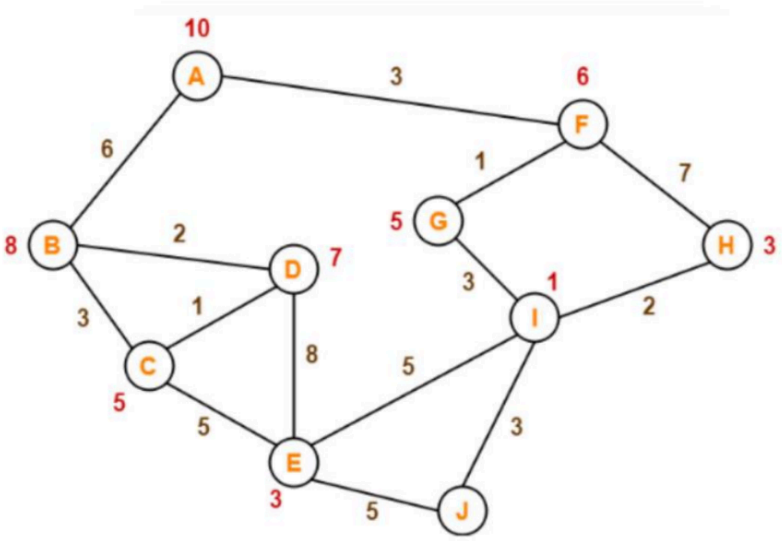
Artificial Intelligence Lab

Darain shahedi assignment 7

l21ma010

1. Implement A* algorithm in python. Solve following problem as shown in given figure. Find the shortest path from A to J.



Solution:
Problem statement : Shortest path finding using A* algorithm
Given a graph consisting of nodes and edges, where each edge has an associated cost, and a heuristic that estimates the cost from each node to goal node using the A* algorithm.

Input:

1) Graph : A set of nodes and edges, where each edge has a non negative cost.
2) Start Node: The node form which the search begins
3) Goal Node: The destination node that we want to reach.
4) Heuristic Function: A function that estimates the cost from each node to the goal node.

Output:

1) Shortest Path: A sequence of nodes representing the shortest path from the start node to the goal node.
2) Path Cost: The total cost of traversing the shortest path.

Constrain:

1) The graph may be directed or undirected.
2) All edge costs are non- negative.
3) The heuristic function is admissible i.e., it never overestimates the cost to reach the goal node.

Algorithm:

1) Initialize the start node with a cost of 0 and calculate its heuristic value.
2) Add the start node to the open set.
3) While the open set is not empty:

   Select the node with the lowest total cost(f-score)from the open set.

   If the selected node is the goal node, reconstruct and return the path from

   The start node to the goal node.

   Otherwise expand the selected node by considering its neighbors and update their cost if a shorter path is found.
4) If no path is found or the open set becomes empty, return failure.

Code:

main.py

Save Run

```python
import heapq

class Node:
    def __init__(self, name, heuristic):
        self.name = name
        self.heuristic = heuristic
        self.neighbors = {}
        self.parent = None
        self.g_cost = float('inf')
        self.f_cost = float('inf')

    def add_neighbor(self, neighbor, distance):
        self.neighbors[neighbor] = distance

def astar(start, goal):
    open_set = []
    closed_set = set()

    start.g_cost = 0
    start.f_cost = start.heuristic

    heapq.heappush(open_set, (start.f_cost, start))

    while open_set:
        current_f_cost, current_node = heapq.heappop(open_set)

        if current_node == goal:
            path = []
            while current_node is not None:
```

main.py

Save Run

```
    while open_set:
        current_f_cost, current_node = heapq.heappop(open_set)

        if current_node == goal:
            path = []
            while current_node is not None:
                path.insert(0, current_node.name)
                current_node = current_node.parent
            return path

        closed_set.add(current_node)

        for neighbor, distance in current_node.neighbors.items():
            if neighbor in closed_set:
                continue

            tentative_g_cost = current_node.g_cost + distance

            if tentative_g_cost < neighbor.g_cost:
                neighbor.parent = current_node
                neighbor.g_cost = tentative_g_cost
                neighbor.f_cost = neighbor.g_cost + neighbor.heuristic

                if neighbor not in open_set:
                    heapq.heappush(open_set, (neighbor.f_cost, neighbor))

    return None

# Define the graph
```

main.py

Save Run

```
49
50      return None
51
52  # Define the graph
53  nodes = {
54      'A': Node('A', 10),
55      'B': Node('B', 8),
56      'C': Node('C', 5),
57      'D': Node('D', 7),
58      'E': Node('E', 3),
59      'F': Node('F', 6),
60      'G': Node('G', 5),
61      'H': Node('H', 3),
62      'I': Node('I', 1),
63      'J': Node('J', 0)
64  }
65
66  nodes['A'].add_neighbor(nodes['B'], 6)
67  nodes['A'].add_neighbor(nodes['F'], 3)
68  nodes['B'].add_neighbor(nodes['D'], 2)
69  nodes['B'].add_neighbor(nodes['C'], 3)
70  nodes['C'].add_neighbor(nodes['E'], 5)
71  nodes['C'].add_neighbor(nodes['D'], 1)
72  nodes['D'].add_neighbor(nodes['E'], 8)
73  nodes['E'].add_neighbor(nodes['I'], 5)
74  nodes['E'].add_neighbor(nodes['J'], 5)
75  nodes['F'].add_neighbor(nodes['G'], 1)
76  nodes['F'].add_neighbor(nodes['H'], 7)
77  nodes['G'].add_neighbor(nodes['I'], 3)
```

```
nodes['H'].add_neighbor(nodes['I'], 2)
nodes['I'].add_neighbor(nodes['J'], 3)

# Find the shortest path from A to J
start_node = nodes['A']
goal_node = nodes['J']
shortest_path = astar(start_node, goal_node)

if shortest_path:
    print("Shortest path from A to J:", shortest_path)
else:
    print("No path found from A to J")

```

Output: