Task 7

Explanation

A Search:

A* is an algorithm used to find the shortest path from a starting point to a goal in a weighted graph. It uses two key values:

1. G(n) → Cost from the start node to the current node.
2. H(n) → Estimated cost (heuristic) from the current node to the goal.

It chooses paths by minimizing:

F(n) = g(n) + h(n)

Where:

G(n) is the known cost from start to current node.

H(n) is an estimate of the cost to reach the goal.

1. Creating the Graph

The Graph class stores:

Self.graph: A dictionary to store nodes and their neighbors with costs.

Self.h: A dictionary for heuristic values (estimated distances to goal).

2. Adding Edges

This function adds connections between nodes.

```
Def add_edge(self, node, neighbor, cost):
  If node not in self.graph:
    Self.graph[node] = []
  Self.graph[node].append((neighbor, cost))
```

example, calling

g.add_edge(‘A’, ‘B’, 1)

Means A is connected to B with a cost of 1

3. Setting Heuristic Values

This function stores the heuristic values.

```
Def set_heuristic(self, heuristic):
   Self.h = heuristic
Heuristic = {'A': 5, 'B': 3, 'C': 2, 'D': 1, 'E': 0}
```

4. A Search Algorithm*

Initialization

```
Open_list = PriorityQueue()  # Priority queue for nodes
Open_list.put((0, start))  # Add start node with priority 0
G_costs = {start: 0}  # Start node cost is 0
Came_from = {}  # Dictionary to track the path
```

Keeps track of the cost from the start node (g_costs).

Stores which node led to another (came_from).

Exploring Nodes

```
While not open_list.empty():
   Current_cost, current_node = open_list.get()
```

The node with the lowest f(n) cost is picked first.

Checking Goal

```
If current_node == goal:
   # Reconstruct the pa
```

Expanding Neighbors

```
For neighbor, cost in self.graph.get(current_node, []):
   New_cost = g_costs[current_node] + cost
```

Each neighbor's cost is calculated.Updating Costs and Priority

```
If neighbor not in g_costs or new_cost < g_costs[neighbor]:
   G_costs[neighbor] = new_cost
```

```
Priority = new_cost + self.h.get(neighbor, 0)

Open_list.put((priority, neighbor))

Came_from[neighbor] = current_node
```