Neha Dattatray Bhoite

Assignment No 2

Input -

```
import heapq

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v, cost):
        if u not in self.graph:
            self.graph[u] = []
        self.graph[u].append((v, cost))

    def a_star_search(self, start, goal, heuristic):
        open_list = []
        heapq.heappush(open_list, (0 + heuristic[start], 0, start, []))
        closed_set = set()

        while open_list:
            _, cost, node, path = heapq.heappop(open_list)

            if node in closed_set:
                continue

            path = path + [node]
            closed_set.add(node)

            if node == goal:
                return path

            for neighbor, move_cost in self.graph.get(node, []):
                if neighbor not in closed_set:
                    heapq.heappush(open_list, (cost + move_cost + heuristic[neighbor],
cost + move_cost, neighbor, path))

        return None

# Example usage:
g = Graph()
g.add_edge('A', 'B', 1)
```

```
g.add_edge('A', 'C', 4)
g.add_edge('B', 'C', 2)
g.add_edge('B', 'D', 5)
g.add_edge('C', 'D', 1)

g_heuristic = {'A': 7, 'B': 6, 'C': 2, 'D': 0}

print("A* search from A to D:")
print(g.a_star_search('A', 'D', g_heuristic))
```

Output-

```
A* search from A to D:
['A', 'C', 'D']

=== Code Execution Successful ===
```