BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India

DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: Artificial Intelligence and Machine Learning

Name	Manish Shashikant Jadhav
UID no.	2023301005

Experiment 3

AIM :

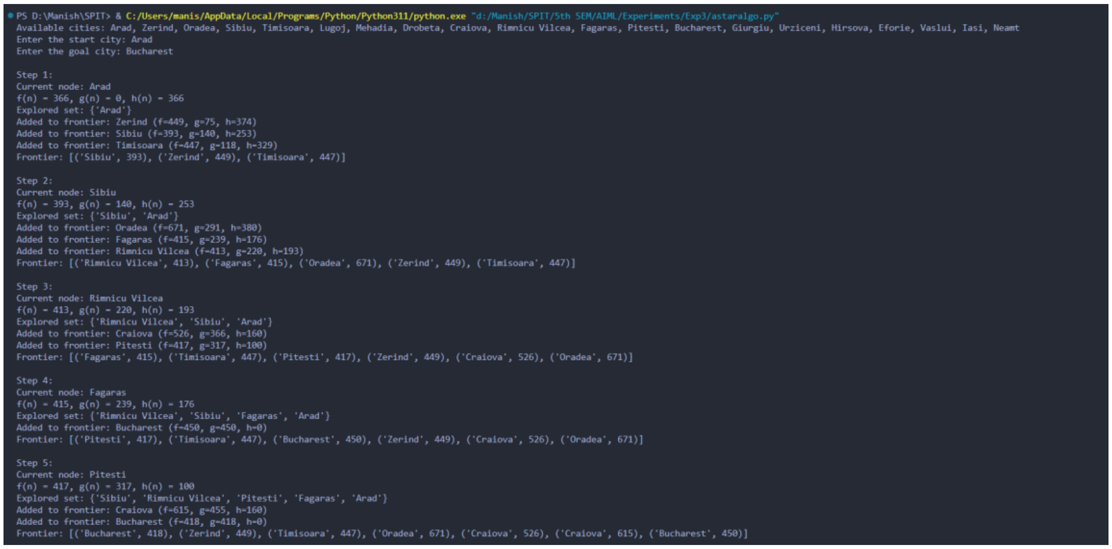
Implement a the Road Map problem using the Informed searching technique using A* search. The start node is Arad and Goal Node is Bucharest. Analyze the algorithm with respect to Completeness, Optimality, time and space Complexity.

CODE:

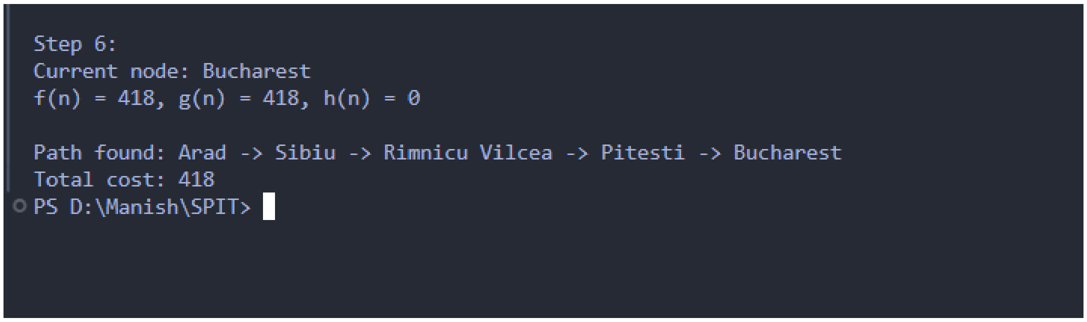
```
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f

def astar_search(graph, start, goal, heuristic):
    start_node = Node(start, None, 0, heuristic[start])
    frontier = []
    heapq.heappush(frontier, start_node)
    explored = set()
    step = 1

    while frontier:
        current_node = heapq.heappop(frontier)

        print(f"\nStep {step}:")
        print(f"Current node: {current_node.state}")
        print(f"f(n) = {current_node.f}, g(n) = {current_node.g}, h(n) = {current_node.h}")
        if current_node.state == goal:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return list(reversed(path))

        explored.add(current_node.state)
        print(f"Explored set: {explored}")

        for neighbor, cost in graph[current_node.state].items():
            if neighbor not in explored:
                g = current_node.g + cost
                h = heuristic[neighbor]
                new_node = Node(neighbor, current_node, g, h)

                if new_node not in frontier:
                    heapq.heappush(frontier, new_node)
                    print(f"Added to frontier: {neighbor} (f={new_node.f}, g={g}, h={h})")
                else:
                    for i, node in enumerate(frontier):
                        if node.state == neighbor and node.g > g:
                            frontier[i] = new_node
                            heapq.heapify(frontier)
                            print(f"Updated in frontier: {neighbor} (f={new_node.f}, g={g}, h={h})")
                            break

        print("Frontier:", [(node.state, node.f) for node in frontier])
        step += 1

    return None

# Romania map values
romania_map = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},
    'Giurgiu': {'Bucharest': 90},
    'Urziceni': {'Bucharest': 85, 'Vaslui': 142, 'Hirsova': 98},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Eforie': {'Hirsova': 86},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}

# heuristic values
heuristic = {
    'Arad': 366, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242, 'Eforie': 161,
    'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,
    'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu Vilcea': 193,
    'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374
}

#Taking user input for start and goal cities
print("Available cities:", ', '.join(romania_map.keys()))
start = input("Enter the start city: ")
goal = input("Enter the goal city: ")

if start not in romania_map or goal not in romania_map:
    print("Invalid start or goal city. Please choose from the available cities.")
else:
    path = astar_search(romania_map, start, goal, heuristic)
    if path:
        print(f"\nPath found: {' -> '.join(path)}")
        print(f"Total cost: {sum(romania_map[path[i]][path[i+1]] for i in range(len(path)-1))}")
```

```
else:
    print("No path found")
```

OUTPUT:



```
PS D:\Manish\SPIT> & C:/Users/manis/AppData/Local/Programs/Python/Python311/python.exe "d:/Manish/SPIT/5th SEM/AIML/Experiments/Exp3/astaralgo.py"
Available cities: Arad, Zerind, Oradea, Sibiu, Timisoara, Lugoj, Mehadia, Drobeta, Craiova, Rimnicu Vilcea, Fagaras, Pitesti, Bucharest, Giurgiu, Urziceni, Hirsova, Eforie, Vaslui, Iasi, Neamt
Enter the start city: Arad
Enter the goal city: Bucharest

Step 1:
Current node: Arad
f(n) = 366, g(n) = 0, h(n) = 366
Explored set: {'Arad'}
Added to frontier: Zerind (f=449, g=75, h=374)
Added to frontier: Sibiu (f=393, g=140, h=253)
Added to frontier: Timisoara (f=447, g=118, h=329)
Frontier: [('Sibiu', 393), ('Zerind', 449), ('Timisoara', 447)]

Step 2:
Current node: Sibiu
f(n) = 393, g(n) = 140, h(n) = 253
Explored set: {'Sibiu', 'Arad'}
Added to frontier: Oradea (f=671, g=291, h=380)
Added to frontier: Fagaras (f=415, g=239, h=176)
Added to frontier: Rimnicu Vilcea (f=413, g=220, h=193)
Frontier: [('Rimnicu Vilcea', 413), ('Fagaras', 415), ('Oradea', 671), ('Zerind', 449), ('Timisoara', 447)]

Step 3:
Current node: Rimnicu Vilcea
f(n) = 413, g(n) = 220, h(n) = 193
Explored set: {'Rimnicu Vilcea', 'Sibiu', 'Arad'}
Added to frontier: Craiova (f=526, g=366, h=160)
Added to frontier: Pitesti (f=417, g=317, h=100)
Frontier: [('Fagaras', 415), ('Timisoara', 447), ('Pitesti', 417), ('Zerind', 449), ('Craiova', 526), ('Oradea', 671)]

Step 4:
Current node: Fagaras
f(n) = 415, g(n) = 239, h(n) = 176
Explored set: {'Rimnicu Vilcea', 'Sibiu', 'Fagaras', 'Arad'}
Added to frontier: Bucharest (f=450, g=450, h=0)
Frontier: [('Pitesti', 417), ('Timisoara', 447), ('Bucharest', 450), ('Zerind', 449), ('Craiova', 526), ('Oradea', 671)]

Step 5:
Current node: Pitesti
f(n) = 417, g(n) = 317, h(n) = 100
Explored set: {'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Fagaras', 'Arad'}
Added to frontier: Craiova (f=615, g=455, h=160)
Added to frontier: Bucharest (f=418, g=418, h=0)
Frontier: [('Bucharest', 418), ('Zerind', 449), ('Timisoara', 447), ('Oradea', 671), ('Craiova', 526), ('Craiova', 615), ('Bucharest', 450)]
```



```
Step 6:
Current node: Bucharest
f(n) = 418, g(n) = 418, h(n) = 0

Path found: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
Total cost: 418
PS D:\Manish\SPIT>
```

Analysis of Algorithm

1. **Completeness:** A* search is complete, meaning it will always find a solution if one exists, provided that:

a) The branching factor is finite
b) All edge costs are positive
c) The heuristic function is admissible (never overestimates the cost to the goal)

2. **Optimality:** A* search is optimal if the heuristic function is admissible and consistent (monotonic). This means it will always find the least-cost path to the goal if such a path exists. For the Romania map problem, if the heuristic values provided are admissible and consistent, the algorithm will find the optimal path from Arad to Bucharest.

3. **Time Complexity:** The time complexity of A* search depends on the heuristic function. In the worst case, when the heuristic is poor, the time complexity can be exponential, O(b^d), where b is the branching factor and d is the depth of the

	solution. For the Romania map problem, the branching factor is relatively small, and with a good heuristic (like straight-line distance to Bucharest), the algorithm should perform well, exploring significantly fewer nodes than uninformed search methods. 4. **Space Complexity:** The space complexity of A* search is O(b^d), as it needs to store all generated nodes in memory. This is because A* keeps track of the frontier (nodes to be explored) and the explored set. For the Romania map problem, the space requirements are manageable due to the limited number of cities. However, for larger graphs, space can become a limiting factor.
CONCLUSION:	Hence by completing this experiment I came to know about Implementation a the Road Map problem using the Informed searching technique using A* search.