Task 1: A* Search

Objective:

Implement A* search to find the shortest path or optimal solution from a start node to a goal node in a weighted graph.

Task Name:

Implement A* search to find an optimal path in between Arad and Bucharest in the given figure.

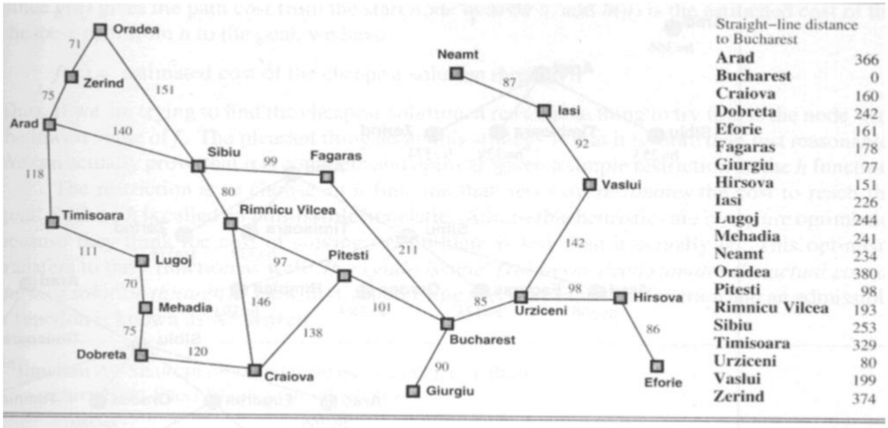

Figure 4.2 Map of Romania with road distances in km, and straight-line distances to Bucharest.

Code

```
import heapq

graph={}

graph = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    '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},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    '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, 'Hirsova': 98, 'Vaslui': 142},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Eforie': {'Hirsova': 86},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}
```

```
heuristic = {
    'Arad': 366, 'Zerind': 374, 'Oradea': 380, 'Timisoara': 329,
    'Lugoj': 244, 'Mehadia': 241, 'Drobeta': 242, 'Craiova': 160,
    'Sibiu': 253, 'Rimnicu Vilcea': 193, 'Fagaras': 178, 'Pitesti': 98,
    'Bucharest': 0, 'Giurgiu': 77, 'Urziceni': 80, 'Hirsova': 151,
    'Eforie': 161, 'Vaslui': 199, 'Iasi': 226, 'Neamt': 234
}


def a_star_search(graph,start,goal):
    list=[]
    heapq.heappush(list,(0+heuristic[start],0,start,[]))

    visited_list=set()
    while list:
        _,cost,current_node,path=heapq.heappop(list)

        if current_node in visited_list:
            continue
        visited_list.add(current_node)
        path=path+[current_node]

        if current_node == goal:
            return path, cost

        for neighbor, distance in graph[current_node].items():
            if neighbor not in visited_list:
                total_cost = cost + distance
                heapq.heappush(list, (total_cost + heuristic[neighbor], total_cost, neighbor, path))

    return None, None

# Execute A* search from Arad to Bucharest
path, cost = a_star_search(graph, 'Oradea', 'Hirsova')

print("Optimal Path:", path)
print("Total Cost:", cost)
```

Output

```
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Optimal Path: ['Zerind', 'Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Total Cost: 493
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Optimal Path: ['Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Total Cost: 429
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Optimal Path: ['Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest', 'Urziceni', 'Hirsova']
Total Cost: 612
PS C:\Users\Lenovo\Desktop\python>
```

Task 2: Greedy Best First Search

Objective:

Implement a Greedy Best First search function to find a path to the goal node in a graph or a search space that minimizes the estimated cost to the goal at each step

Task Name:

Write a program to implement Greedy Best First Search algorithm for above figure to find an optimal path between Arad and Bucharest.

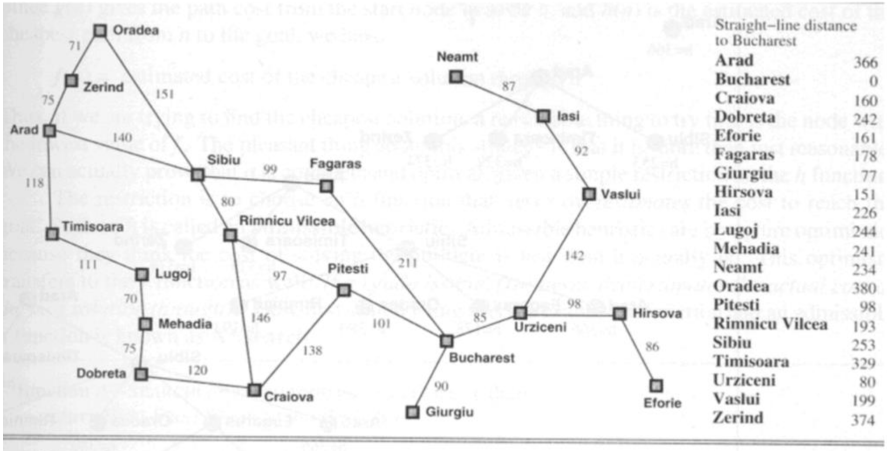

Figure 4.2 Map of Romania with road distances in km, and straight-line distances to Bucharest.

Code:

```
import heapq
graph = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    '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},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    '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, 'Hirsova': 98, 'Vaslui': 142},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Eforie': {'Hirsova': 86},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}
```

```
heuristic = {
    'Arad': 366, 'Zerind': 374, 'Oradea': 380, 'Timisoara': 329,
    'Lugoj': 244, 'Mehadia': 241, 'Drobeta': 242, 'Craiova': 160,
    'Sibiu': 253, 'Rimnicu Vilcea': 193, 'Fagaras': 178, 'Pitesti': 98,
    'Bucharest': 0, 'Giurgiu': 77, 'Urziceni': 80, 'Hirsova': 151,
    'Eforie': 161, 'Vaslui': 199, 'Iasi': 226, 'Neamt': 234
}

def greedy_best_first_search(graph, start, goal):
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start, []))

    closed_list = set()

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

        if current_node in closed_list:
            continue

        closed_list.add(current_node)
        path = path + [current_node]

        if current_node == goal:
            return path

        for neighbor in graph[current_node]:
            if neighbor not in closed_list:
                heapq.heappush(open_list, (heuristic[neighbor], neighbor, path))

    return None

path = greedy_best_first_search(graph, 'Arad', 'Bucharest')

print("Path:", path)
```

Output:

```
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Path: ['Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Bucharest']
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Path: None
PS C:\Users\Lenovo\Desktop\python> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Path: ['Sibiu', 'Fagaras', 'Bucharest', 'Pitesti', 'Craiova', 'Drobeta']
```