# Lab Task: 11

***——————————————786**
**————————————————***

**Name: Muhammad Sherjeel Akhtar**

**Roll No: 20p-0101**

**Subject: Artificial Intelligence Lab**

**Submitted To Respected Ma’am: Hurmat Hidayat**

**Section: BCS-6C**

**------------------------------------------------------------------------------------------------------786**

**Name: Muhammad Sherjeel Akhtar**

**Roll No: 20p-0101**

**Lab Task: 11**

**Subject: Artificial Intelligence Lab**

**Section: BCS-6C**

**Submitted To Respected Ma'am: Miss Hurmat Hidayat**

**----------------------------------------------------------------------------------------------------------**

### Importing Libraries

In [10]:
```
from queue import PriorityQueue
from geopy.distance import geodesic
```

### Importing Libraries

In [10]:
```python
from queue import PriorityQueue
from geopy.distance import geodesic
```

### Defining The Latitude And Longitude Of Each City

In [11]:
```python
lat_long = {
    'Arad': (46.1667, 21.3167), 'Bucharest': (44.4167, 26.1000),
    'Craiova': (44.3333, 23.8167), 'Drobeta': (44.6259, 22.6566),
    'Eforie': (44.0667, 28.6333), 'Fagaras': (45.8416, 24.9730),
    'Giurgiu': (43.9037, 25.9699), 'Hirsova': (44.6833, 27.9500),
    'Iasi': (47.1585, 27.6014), 'Lugoj': (45.6904, 21.9033),
    'Neamt': (46.9283, 26.3705), 'Oradea': (47.0553, 21.9214),
    'Pitesti': (44.8565, 24.8697), 'Rimnicu Vilcea': (45.1042, 24.3758),
    'Sibiu': (45.7977, 24.1521), 'Timisoara': (45.7489, 21.2087),
    'Urziceni': (44.7167, 26.6333), 'Vaslui': (46.6333, 27.7333),
    'Zerind': (46.6225, 21.5174)
}
```

### Define The Graph Via Dictionaries

In [12]:
```python
graph = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Mehadia': 70, 'Timisoara': 111},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
```

### Define The Graph Via Dictionaries

In [12]:
```python
graph = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Eforie': {'Hirsova': 86},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Giurgiu': {'Bucharest': 90},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Iasi': {'Neamt': 87, 'Vaslui': 92},
    'Lugoj': {'Mehadia': 70, 'Timisoara': 111},
    'Neamt': {'Iasi': 87},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Vaslui': {'Iasi': 92, 'Urziceni': 142},
    'Zerind': {'Arad': 75, 'Oradea': 71}
}
```

### Defining Node Via Class

In [13]:
```python
class Node:
    def __init__(self, city, parent=None, g_cost=0, h_cost=0):
        self.city = city
        self.parent = parent
        self.g_cost = g_cost
        self.h_cost = h_cost
        self.f_cost = g_cost + h_cost
```

## Defining Node Via Class

In [13]:
```
class Node:
    def __init__(self, city, parent=None, g_cost=0, h_cost=0):
        self.city = city
        self.parent = parent
        self.g_cost = g_cost
        self.h_cost = h_cost
        self.f_cost = g_cost + h_cost
```

## Shortest Path Function

In [14]:
```
def find_shortest_path(start_city, goal_city):
    open_list = PriorityQueue()
    open_list.put((0, Node(start_city)))
    visited = set()

    while not open_list.empty():
        current_node = open_list.get()[1]
        current_city = current_node.city

        if current_city == goal_city:
            path = []
            distance = current_node.g_cost
            while current_node:
                path.append(current_node.city)
                current_node = current_node.parent
            path.reverse()
            return path, distance

        visited.add(current_city)

        for neighbor, cost in graph[current_city].items():
            if neighbor in visited:
                continue

            g_cost = current_node.g_cost + cost
            h_cost = calculate_heuristic(neighbor, goal_city)
            f_cost = g_cost + h_cost
            neighbor_node = Node(neighbor, current_node, g_cost, h_cost)

            open_list.put((f_cost, neighbor_node))

    return None, float('inf')
```

## Defining Heuristic Function

In [15]:
```
def calculate_heuristic(city, goal_city):
    lat1, long1 = lat_long[city]
    lat2, long2 = lat_long[goal_city]
    distance = geodesic((lat1, long1), (lat2, long2)).km
    return distance
```

## Visualization Of The Test Examples

In [16]:
```
start_city = 'Arad'
goal_city = 'Bucharest'
shortest_path, distance = find_shortest_path(start_city, goal_city)
print(f"Shortest path from {start_city} to {goal_city}: {shortest_path}")
print(f"Distance: {distance}")
```

```
Shortest path from Arad to Bucharest: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Distance: 418
```

## FIN-------

# FIN!!