Lab Task 10:

***_____786**
_____**

Name: Muhammad Sherjeel Akhtar

Roll No: 20p-0101

Subject: Artificial Intelligence Lab

Submitted To Respected
Ma’am: Hurmat Hidayat

Section: BCS-6C

Importing Libraries And Heuristic Function:

----------786

Name: Muhammad Sherjeel Akhtar

Roll No: 20p-0101

Lab Task: 10

Subject: Artificial Intelligence Lab

Section: BCS-6C

Submitted To Respected Ma'am: Miss Hurmat Hidayat

Importing Library

In [15]:
```
import heapq
```


Importing Library

In [15]:
```
import heapq
```


Creating A Heuristic Function To Calculate Straight-Line Distance Between Two Cities

In [16]:
```
def heuristic(city, destination):
    return 0
```

Creating Best_First_Search Function Acquiring Three Parameters:

Graph

Start State

Goal State

In [17]:
```
def best_first_search(graph, start, goal):
    queue = [(heuristic(start, goal), start)]
    visited = set()
    while queue:
        _, current_city = heapq.heappop(queue)
        if current_city == goal:
            return graph[current_city]
        visited.add(current_city)

        for neighbor, distance in graph[current_city]:
            neighbor_city = neighbor
            if neighbor_city not in visited:
                h = heuristic(neighbor_city, goal)
                heapq.heappush(queue, (h, neighbor_city))
    return None
```

Representation Of Graph Given In Lab Manual In A Dictionary (Adjancency List)

In [18]:
```
graph = {
    'Arad': [('Sibiu', 140), ('Timisoara', 118), ('Zerind', 75)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101)],
    'Sibiu': [('Fagaras', 99), ('Rimnicu Vilcea', 80), ('Arad', 140)],
    'Timisoara': [('Lugoj', 111), ('Arad', 118)],
    'Fagaras': [('Bucharest', 211), ('Sibiu', 99)],
    'Pitesti': [('Bucharest', 101)],
    'Rimnicu Vilcea': [('Sibiu', 80), ('Pitesti', 97)],
    'Lugoj': [('Timisoara', 111)],
    'Giurgiu': [('Bucharest', 90)],
    'Zerind': [('Arad', 75), ('Oradea', 71)],
    'Oradea': [('Zerind', 71), ('Sibiu', 151)],
    'Craiova': [('Drobeta', 120), ('Rimnicu Vilcea', 146), ('Pitesti', 138)],
    'Drobeta': [('Craiova', 120), ('Mehadia', 75)],
    'Mehadia': [('Drobeta', 75), ('Lugoj', 70)],
    '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)]
}
```

Setting Start And End States By Setting The Cities In Both Variables

In [19]:
```
start_city = 'Giurgiu'
goal_city = 'Sibiu'
```

Calling The best_first_search Algorithm And Storing It In A Distance Variable

In [20]:
```
distance = best_first_search(graph, start_city, goal_city)
```

Visualizing The Results

In [21]:
```
print(f"Shortest distance from {start_city} to {goal_city}: {distance}")
```

```
Shortest distance from Giurgiu to Sibiu: [('Fagaras', 99), ('Rimnicu Vilcea', 80), ('Arad', 140)]
```

FIN------

FIN!