Name: Jawad Ahmed

Roll no: 20P-0165

LAB TASK 10 (Best First Search)

In [2]:
```python
import networkx as nx
import matplotlib.pyplot as plt
```

Generate Graph for Map of Romania

In [3]:
```python
G = nx.Graph()

G.add_nodes_from(["Arad", "Bucharest", "Oradea", "Zerind", "Timisoara", "Lugoj", "Mehadia", "Dobreta", "Craiova", "Pitesti", "Rimnicu Vilcea", "Sibiu", "Fagarus", "Giurgiu", "Urziceni", "Vaslui", "lasi", "Neamt", "Hirsova", "Efor
```

In [4]:
```python
edges = [("Arad", "Zerind", 75),("Arad", "Sibiu", 140),
         ("Arad", "Timisoara", 118), ("Zerind", "Oradea", 71), ("Timisoara", "Lugoj", 111), ("Lugoj", "Mehadia", 70), ("Mehadia", "Dobreta", 75),
         ("Dobreta", "Craiova", 120), ("Craiova", "Rimnicu Vilcea", 146), ("Craiova", "Pitesti", 138), ("Pitesti", "Rimnicu Vilcea", 97),
         ("Rimnicu Vilcea", "Sibiu", 80), ("Oradea", "Sibiu", 151), ("Sibiu", "Fagaras", 99), ("Fagaras", "Bucharest", 211),
         ("Pitesti", "Bucharest", 101), ("Bucharest", "Giurgiu", 90), ("Bucharest", "Urziceni", 85), ("Urziceni", "Vaslui", 142), ("Vaslui", "lasi", 92),
         ("Lasi", "Neamt", 87), ("Urziceni", "Hirsova", 98), ("Hirsova", "Eforie", 86), ("Fagaras", "Bucharest", 211)] #add remaining edges to the list

for edge in edges:
    G.add_edge(edge[0], edge[1], weight=edge[2])
```

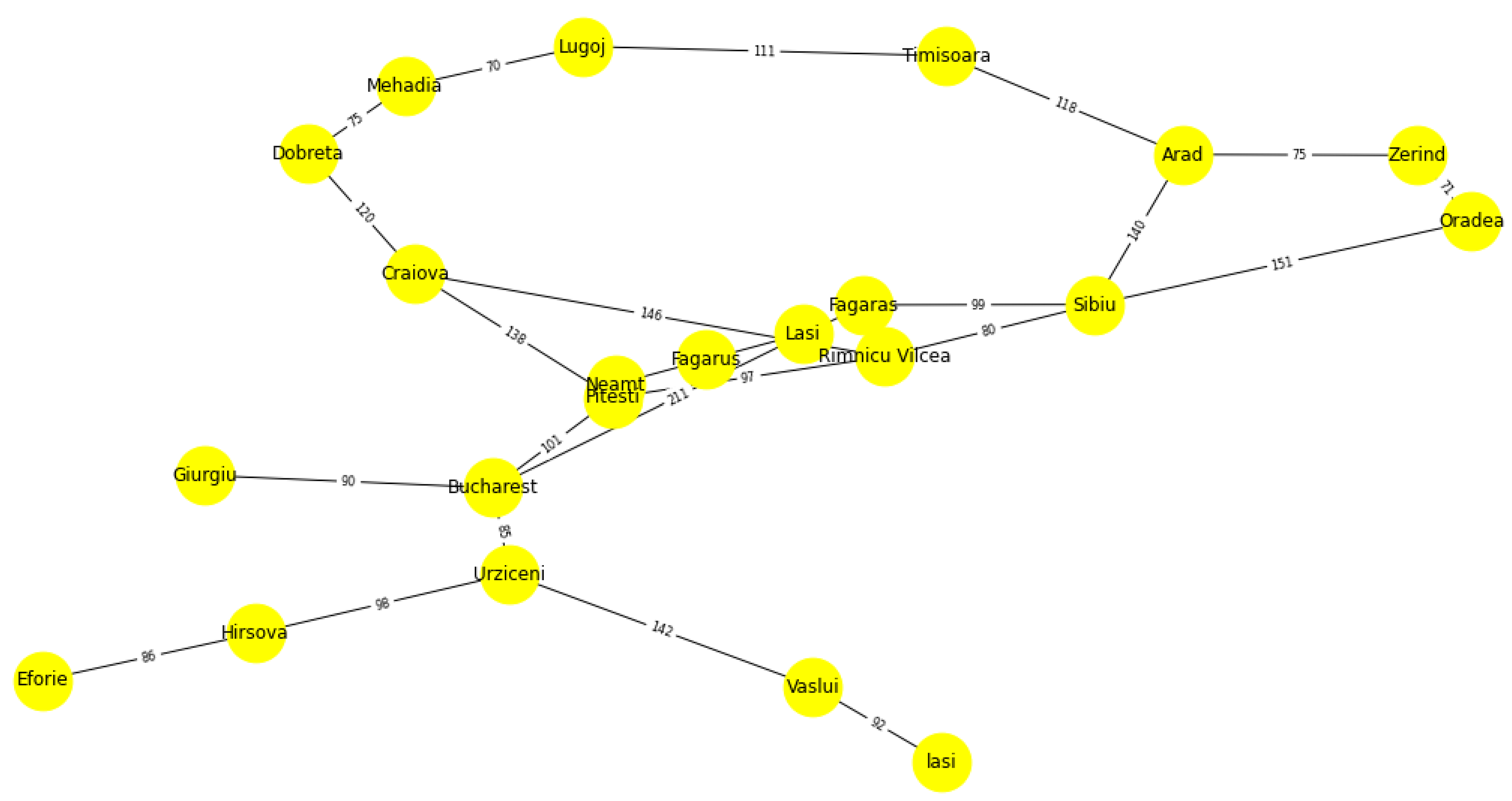
In [5]:
```python
huristic_values = {'Arad': 366, 'Sibiu': 253, 'Bucharest': 0, 'Craiova': 160, 'Eforie': 161, 'Fagaras': 176, 'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu Vilcea': 193, 'Dobreta': 242, 'Hirsova': 151, 'Ias
```

In [6]:
```python
# Set node positions using Kamada-Kawai layout
pos = nx.kamada_kawai_layout(G)
```

In [7]:
```python
# Draw graph with labels and edge weights
plt.figure(figsize=(16, 8))
nx.draw(G, pos, with_labels=True, font_size=12, node_size= 1500, node_color ="yellow")

edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)

plt.show()
```



In [8]:
```python
class Node():
    def __init__(self, state, parent, action):
        self.state = state
        self.parent = parent
        self.action = action
```

In [9]:
```python
class Frontier():
    def __init__(self):
        self.frontier = []

    def empty(self):
        return (len(self.frontier) == 0)

    def add(self, node):
        self.frontier.append(node)

    def remove(self):
        if self.empty():
            return Exception("Empty Queue")
        else:
            node = self.frontier[0]
            self.frontier = self.frontier[1:]
            return node
```

In [10]:
```python
def best_first_search(start, goal, graph):
    frontier = Frontier()
    frontier.add(Node(state=start, parent=None, action=None))

    explored = set()

    while True:
        if frontier.empty():
            return None

        node = frontier.remove()

        if node.state == goal:
            path = []
            while node.parent is not None:
                path.append(node.state)
                node = node.parent
            path.append(start)
            return path[::-1]

#         selection of best first search
        neighbours = list(graph.neighbors(node.state))
        smaller_huristic_node = neighbours[0]
        for neighbour in neighbours:
            if neighbour not in explored:
                if smaller_huristic_node is neighbour:
                    continue
                else:
                    if (huristic_values[neighbour] < huristic_values[smaller_huristic_node]):
                        smaller_huristic_node = neighbour

        explored.add(node.state)
        child = Node(state=smaller_huristic_node, parent=node, action=None)
        frontier.add(child)
```

In [11]:
```python
path = best_first_search('Arad', 'Bucharest', G)
```

In [12]:
```python
def calculate_cost(path, graph):
    total_cost = 0
    for city in range(len(path) - 1):
        edge_data = graph.get_edge_data(path[city], path[city + 1])
        total_cost += edge_data['weight']
    return total_cost
```

In [13]:
```python
cost = calculate_cost(path, G)
```

In [14]:
```python
print("The path will using Best First Search", path)
print("Cost of the path willbe: ", cost)
```

```
The path will using Best First Search ['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
Cost of the path willbe:  450
```

In [ ]:

In [84]:

In [85]:

In [99]:

In [100…

In [101…

In [102…

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

In [ ]:

In [ ]: