


## Lab 08 - Task

### Travelling on vacations to Romania Taha Abbas Ali P200119 BCS-6A

Jupyter TahaAbbasAli\_P200119\_6A\_S23\_AILab\_08 (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Generate graph for Map of Romania

Note: Complete the missing part of the code

```
In [2]: G = nx.Graph()

G.add_nodes_from(["Arad", "Bucharest", "Oradea", "Zerind", "Timisoara", "Lugoj", "Mehadia", "Dobreta", "Craiova",
                  "Rimnicu Vilcea", "Sibiu", "Fagaras", "Pitesti", "Giurgiu", "Urziceni", "Vaslui", "Lasi",
                  "Neamt", "Hirsova", "Eforie"]) #add remaining nodes to the list
```

```
In [3]: edges = [
    ("Arad", "Zerind", 75),
    ("Arad", "Sibiu", 140),
    ("Arad", "Timisoara", 118),
    ("Bucharest", "Urziceni", 85),
    ("Bucharest", "Giurgiu", 90),
    ("Bucharest", "Pitesti", 101),
    ("Bucharest", "Fagaras", 211),
    ("Craiova", "Dobreta", 120),
    ("Craiova", "Pitesti", 138),
    ("Craiova", "Rimnicu Vilcea", 146),
    ("Dobreta", "Mehadia", 75),
    ("Eforie", "Hirsova", 86),
    ("Fagaras", "Sibiu", 99),
    ("Hirsova", "Urziceni", 98),
    ("Lasi", "Neamt", 87),
    ("Lasi", "Vaslui", 92),
    ("Lugoj", "Mehadia", 70),
    ("Lugoj", "Timisoara", 111),
    ("Oradea", "Zerind", 71),
    ("Oradea", "Sibiu", 151),
    ("Pitesti", "Rimnicu Vilcea", 97),
    ("Rimnicu Vilcea", "Sibiu", 80),
    ("Urziceni", "Vaslui", 142)
]

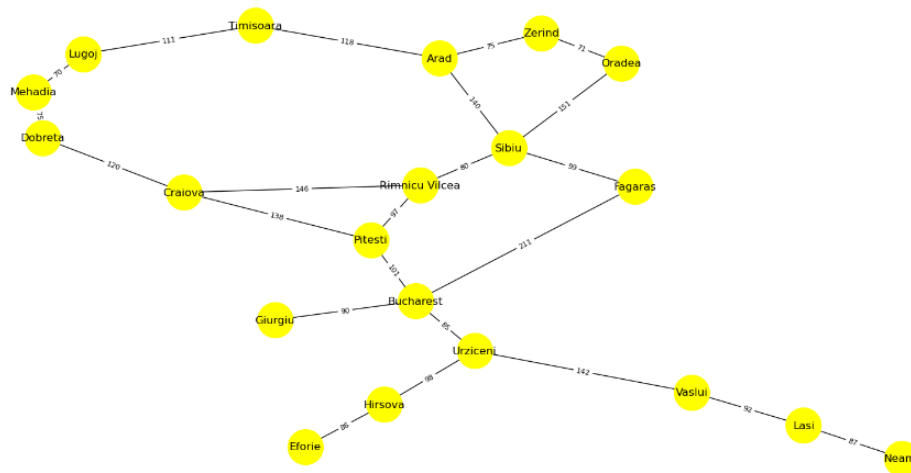
for edge in edges:
    G.add_edge(edge[0], edge[1], weight=edge[2])
```

```
In [4]: # Set node positions using Kamada-Kawai layout
pos = nx.kamada_kawai_layout(G)
```

```
In [5]: # 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 [9]: def bfs_search(graph, start, goal):
        """Perform breadth-first search from start to goal."""
        #your code
        frontier = QueueFrontier()          # initializing Queue
        start_node = Node(start, None, None) # initializing Node
        frontier.add(start_node)
        explored = set()

        while not frontier.empty(): # loop executes till Queue frontier is not empty
            #according to given algorithm
            node = frontier.remove()

            if node.state == goal:
                actions = []
                total_nodes = []

                while node.parent is not None:
                    actions.append({"weight": graph[node.state][node.parent.state]["weight"]}) # creates of list of
                    total_nodes.append(node.state)
                    node = node.parent

                # reverse the final result sets
                actions.reverse()
                total_nodes.reverse()
                return actions, total_nodes

            explored.add(node.state)

        for adjacent in graph.neighbors(node.state):
            if adjacent not in explored and not frontier.contains_state(adjacent): # calling contain_state funct.
                child_node = Node(adjacent, node, None)
                frontier.add(child_node)

        return None
```

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

Run

```
In [10]: def dfs_search(graph, start, goal):
    """Perform depth-first search from start to goal."""
    #your code

    frontier = StackFrontier()           # initializing Stack
    start_node = Node(start, None, None) # initializing Node
    frontier.add(start_node)
    explored = set()

    while not frontier.empty():           # loop executes till Queue frontier is not empty
        #according to given algorithm
        node = frontier.remove()

        if node.state == goal:
            actions = []
            total_nodes = []
            while node.parent is not None:
                actions.append({"weight": G[node.state][node.parent.state]["weight"]}) # creates of list of dict
                total_nodes.append(node.state)
                node = node.parent

            # reverse the final result sets
            actions.reverse()
            nodes.reverse()
            return actions, total_nodes

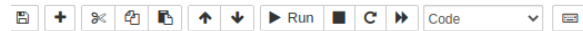
        explored.add(node.state)

        for adjacent in G.neighbors(node.state):
            if adjacent not in explored and not frontier.contains_state(adjacent): # calling contain_state function
                child_node = Node(adjacent, node, None)
                frontier.add(child_node)

    return None
```

```
In [11]: # Perform a BFS search from Arad to Bucharest
actions, nodes = bfs_search(G, 'Arad', 'Bucharest')
print('BFS path from Arad to Bucharest:')
print(actions)
print(nodes)
```

```
BFS path from Arad to Bucharest:
[{'weight': 140}, {'weight': 99}, {'weight': 211}]
['Sibiu', 'Fagaras', 'Bucharest']
```



```
return None
```

```
In [11]: # Perform a BFS search from Arad to Bucharest
actions, nodes = bfs_search(G, 'Arad', 'Bucharest')
print('BFS path from Arad to Bucharest:')
print(actions)
print(nodes)
```

```
BFS path from Arad to Bucharest:
[{'weight': 140}, {'weight': 99}, {'weight': 211}]
['Sibiu', 'Fagaras', 'Bucharest']
```

```
In [12]: # Perform a DFS search from Arad to Bucharest
actions, nodes = dfs_search(G, 'Arad', 'Bucharest')
print('DFS path from Arad to Bucharest:')
print(actions)
print(nodes)
```

```
DFS path from Arad to Bucharest:
[{'weight': 118}, {'weight': 111}, {'weight': 70}, {'weight': 75}, {'weight': 120}, {'weight': 138}, {'weight': 101}]
['Bucharest', 'Pitesti', 'Craiova', 'Dobreta', 'Mehadia', 'Lugoj', 'Timisoara']
```

### Sample output

BFS path from Arad to Bucharest:

```
[{'weight': 140}, {'weight': 99}, {'weight': 211}]
```

```
['Sibiu', 'Fagaras', 'Bucharest']
```

DFS path from Arad to Bucharest:

```
[{'weight': 118}, {'weight': 111}, {'weight': 70}, {'weight': 75}, {'weight': 120}, {'weight': 138}, {'weight': 101}]
```

```
['Timisoara', 'Lugoj', 'Mehadia', 'Dobreta', 'Craiova', 'Pitesti', 'Bucharest']
```

In [ ]: