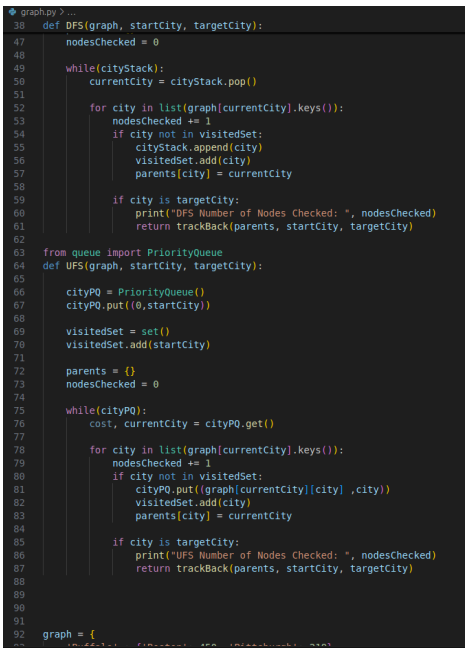
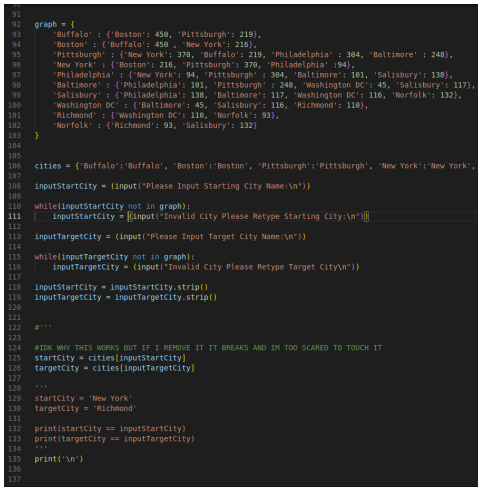
I designed my algorithm to be extraordinary modular therefore reducing the amount of work needed in order to code each algorithm. This is because by having different data structures that choose which nodes should be analyzed allows for each of these algorithms to turn into one another. Since stack will choose the most recent in it will simulate DFS where it will shoot down paths as it keeps choosing the first node it sees. Queues work like BFS where it continuously keeps checking the first things it saw before going to next one. While priority queues can operate like UFS if you set it to cost as priority since the lowest cost will always be next analyzed. I found it also necessary how each node got found in order to be able to back track the path as required by the assignment. This algorithm likely has room for improvement here as it is O(n) in order to keep get back the path and I imagine could've been done alongside the search algorithm but IDK how. The algorithm seems to have some struggle with direct assignment over taking from input for whatever reason when I took input from user even if it were identical to the string if it were hard coded the algorithms would break. I couldn't find the underlying cause of this problem and managed to get around by having a map that assigns a hardcoded value corresponding to the users input.

Code Screenshots:

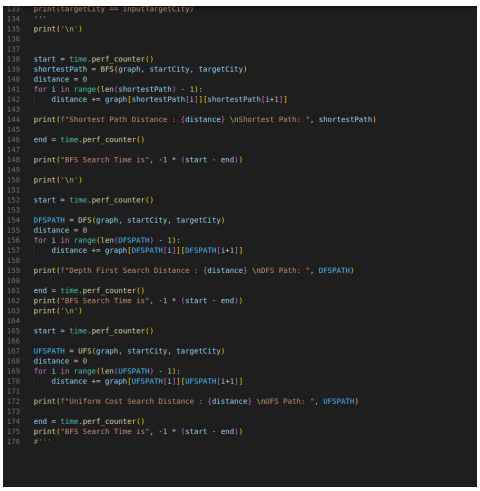
```python
import time
def trackBack(parents, startCity, targetCity):
    currentCity = targetCity
    path = []
    while currentCity is not startCity:
        path.insert(0,currentCity)
        currentCity = parents[currentCity]
    path.insert(0,currentCity)
    return path

def BFS(graph, startCity, targetCity):

    cityQueue = []
    cityQueue.append(startCity)

    visitedSet = set()
    visitedSet.add(startCity)

    parents = {}
    nodesChecked = 0

    while(cityQueue):
        currentCity = cityQueue.pop(0)

        for city in list(graph[currentCity].keys()):
            nodesChecked += 1
            if city not in visitedSet:
                cityQueue.append(city)
                visitedSet.add(city)
                parents[city] = currentCity

            if city is targetCity:
                print("BFS Number of Nodes Checked: ", nodesChecked)
                return trackBack(parents, startCity, targetCity)


def DFS(graph, startCity, targetCity):

    cityStack = []
    cityStack.append(startCity)

    visitedSet = set()
    visitedSet.add(startCity)

    parents = {}
    nodesChecked = 0
```



```python
def DFS(graph, startCity, targetCity):
    nodesChecked = 0

    while(cityStack):
        currentCity = cityStack.pop()

        for city in list(graph[currentCity].keys()):
            nodesChecked += 1
            if city not in visitedSet:
                cityStack.append(city)
                visitedSet.add(city)
                parents[city] = currentCity

            if city is targetCity:
                print("DFS Number of Nodes Checked: ", nodesChecked)
                return trackBack(parents, startCity, targetCity)

from queue import PriorityQueue
def UFS(graph, startCity, targetCity):

    cityPQ = PriorityQueue()
    cityPQ.put((0,startCity))

    visitedSet = set()
    visitedSet.add(startCity)

    parents = {}
    nodesChecked = 0

    while(cityPQ):
        cost, currentCity = cityPQ.get()

        for city in list(graph[currentCity].keys()):
            nodesChecked += 1
            if city not in visitedSet:
                cityPQ.put((graph[currentCity][city] ,city))
                visitedSet.add(city)
                parents[city] = currentCity

            if city is targetCity:
                print("UFS Number of Nodes Checked: ", nodesChecked)
                return trackBack(parents, startCity, targetCity)



graph = {
```



```python
graph = {
    'Buffalo' : {'Boston': 450, 'Pittsburgh': 219},
    'Boston' : {'Buffalo': 450 , 'New York': 216},
    'Pittsburgh' : {'New York': 379, 'Buffalo': 219, 'Philadelphia' : 304, 'Baltimore' : 248},
    'New York' : {'Boston': 216, 'Pittsburgh': 370, 'Philadelphia' :94},
    'Philadelphia' : {'New York': 94, 'Pittsburgh' : 304, 'Baltimore': 101, 'Salisbury': 138},
    'Baltimore' : {'Philadelphia': 101, 'Pittsburgh' : 248, 'Washington DC': 45, 'Salisbury': 117},
    'Salisbury' : {'Philadelphia': 138, 'Baltimore': 117, 'Washington DC': 116, 'Norfolk': 132},
    'Washington DC' : {'Baltimore': 45, 'Salisbury': 116, 'Richmond': 110},
    'Richmond' : {'Washington DC': 110, 'Norfolk': 93},
    'Norfolk' : {'Richmond': 93, 'Salisbury': 132}
}


cities = {'Buffalo':'Buffalo', 'Boston':'Boston', 'Pittsburgh':'Pittsburgh', 'New York':'New York',

inputStartCity = (input("Please Input Starting City Name:\n"))

while(inputStartCity not in graph):
    inputStartCity = (input("Invalid City Please Retype Starting City:\n"))

inputTargetCity = (input("Please Input Target City Name:\n"))

while(inputTargetCity not in graph):
    inputTargetCity = (input("Invalid City Please Retype Target City\n"))

inputStartCity = inputStartCity.strip()
inputTargetCity = inputTargetCity.strip()


#'''

#IDK WHY THIS WORKS BUT IF I REMOVE IT IT BREAKS AND IM TOO SCARED TO TOUCH IT
startCity = cities[inputStartCity]
targetCity = cities[inputTargetCity]
'''
startCity = 'New York'
targetCity = 'Richmond'

print(startCity == inputStartCity)
print(targetCity == inputTargetCity)
'''
print('\n')
```



```python
print(targetCity == inputTargetCity)
'''
print('\n')


start = time.perf_counter()
shortestPath = BFS(graph, startCity, targetCity)
distance = 0
for i in range(len(shortestPath) - 1):
    distance += graph[shortestPath[i]][shortestPath[i+1]]

print(f"Shortest Path Distance : {distance} \nShortest Path: ", shortestPath)

end = time.perf_counter()

print("BFS Search Time is", -1 * (start - end))

print('\n')

start = time.perf_counter()

DFSPATH = DFS(graph, startCity, targetCity)
distance = 0
for i in range(len(DFSPATH) - 1):
    distance += graph[DFSPATH[i]][DFSPATH[i+1]]

print(f"Depth First Search Distance : {distance} \nDFS Path: ", DFSPATH)

end = time.perf_counter()
print("BFS Search Time is", -1 * (start - end))
print('\n')

start = time.perf_counter()

UFSPATH = UFS(graph, startCity, targetCity)
distance = 0
for i in range(len(UFSPATH) - 1):
    distance += graph[UFSPATH[i]][UFSPATH[i+1]]

print(f"Uniform Cost Search Distance : {distance} \nUFS Path: ", UFSPATH)

end = time.perf_counter()
print("BFS Search Time is", -1 * (start - end))
#'''
```

Output:

```
omniladder:~/Codes_for_School/COSC411/Project1$ python graph.py
Please Input Starting City Name:
New York
Please Input Target City Name:
Richmond


BFS Number of Nodes Checked:  26
Shortest Path Distance : 773
Shortest Path:  ['New York', 'Pittsburgh', 'Baltimore', 'Washington DC', 'Richmond']
BFS Search Time is 0.00010180399840464815


DFS Number of Nodes Checked:  12
Depth First Search Distance : 457
DFS Path:  ['New York', 'Philadelphia', 'Salisbury', 'Norfolk', 'Richmond']
BFS Search Time is 4.063700180267915e-05


UFS Number of Nodes Checked:  14
Uniform Cost Search Distance : 350
UFS Path:  ['New York', 'Philadelphia', 'Baltimore', 'Washington DC', 'Richmond']
BFS Search Time is 0.00013104900426696986
```