Artificial Intelligence Assignment 1:

Nu- Id: K163989

Name: Safeer Ahmed

Section: B

Data Representation:

```
graph1 = {
    'Arad' : [['Zerind',75],['Timisoara',118],['Sibiu',140]],
    'Bucharest' : [['Urziceni',85],['Pitesti',101],['Fagaras',211],['Giurgiu',90]],
    'Craiova' : [['Pitesti',138],['Rimnicu Vilcea',146],['Drobeta',120]],
    'Drobeta' : [['Mehadia',75],['Craiova',120]],
    'Eforie' : [['Hirsova',86]],
    'Fagaras' : [['Sibiu',99],['Bucharest',211]],
    'Giurgiu' : [['Bucharest',90]],
    'Hirsova' : [['Eforie',86],['Urziceni',98]],
    'Iasi' : [['Vaslui',92],['Neamt',87]],
    'Lugoj' : [['Mehadia',70],['Timisoara',111]],
    'Mehadia' : [['Lugoj',70],['Drobeta',75]],
    'Neamt' : [['Iasi',87]],
    'Oradea' : [['Zerind',71],['Sibiu',151]],
    'Pitesti' : [['Rimnicu Vilcea',97],['Craiova',138],['Bucharest',101]],
    'Rimnicu Vilcea' : [['Craiova',146],['Pitesti',97],['Sibiu',80]],
    'Sibiu' : [['Arad',140],['Rimnicu Vilcea',80],['Oradea',151],['Fagaras',99]],
    'Timisoara' : [['Arad',118],['Lugoj',111]],
    'Urziceni' : [['Bucharest',85],['Hirsova',98]],
    'Vaslui' : [['Iasi',92],['Urziceni',142]],
    'Zerind' : [['Arad',75],['Oradea',71]]
}
heuristics= {
    'Arad' : 366,
    'Bucharest' : 0,
    'Craiova' : 160,
    'Drobeta' : 242,
    'Eforie' : 161,
    'Fagaras' : 176,
    'Giurgiu' : 77,
    'Hirsova' : 151,
    'Iasi' : 226,
    'Lugoj' : 244,
    'Mehadia' : 241,
    'Neamt' : 234,
    'Oradea' : 380,
    'Pitesti' : 100,
    'Rimnicu Vilcea' : 193,
    'Sibiu' : 253,
    'Timisoara' : 329,
    'Urziceni' : 80,
    'Vaslui' : 199,
    'Zerind' : 374
}
```

A* Algorithm:

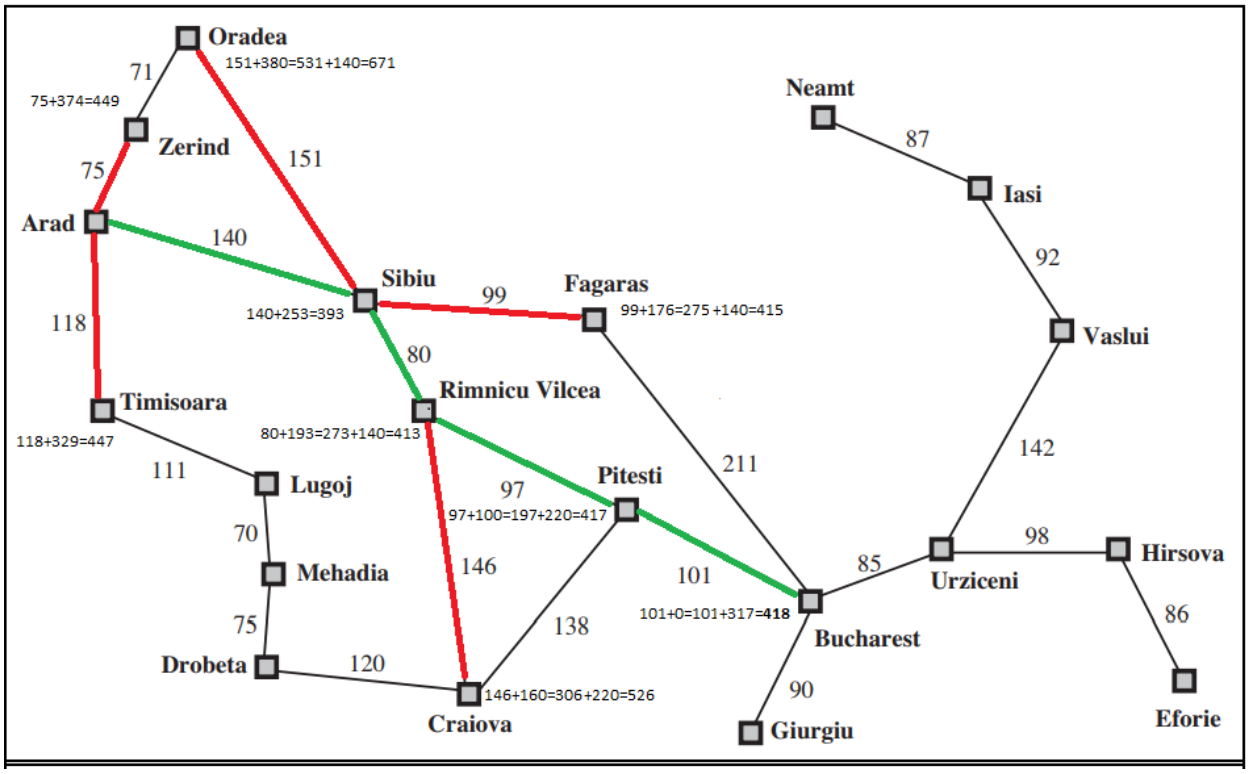
Implementation in Python:

```
def aStar1(graph1,heuristics, start, goal,cost,visited):
    min= heuristics[graph1[start][0][0]]+graph1[start][0][1]
    temp = ''
    visited=visited+[start]
    mincost=0
    while start!=goal:
        for v in graph1[start]:
            if heuristics[v[0]]+v[1]<=min:
                min=heuristics[v[0]]+v[1]
                temp = v[0]
                mincost=v[1]
        cost=cost+mincost
        visited=visited+[temp]
        start = temp
    print(visited)
    print('Cost: '+str(cost))
```

Output:

['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Cost: 418

Visual Representation:



Depth first Search:

Algorithm:

The algorithm recursively calls the function in order to find all the possible paths and their cost from Arad to Bucharest.

Implementation in Python:

```
def dfsrec(graph1,start,goal,path,visited,cost):
    for v in graph1[start]:
        if start==goal:
            print(path)
            print('Cost: '+str(cost))
            return
        if v[0] not in visited:
            dfsrec(graph1,v[0],goal,path+[v[0]],visited+[v[0]],cost+v[1])

dfsrec(graph1,'Arad','Bucharest',['Arad'],['Arad'],0)
```

Output:

```
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Craiova', 'Pitesti', 'Bucharest']
Cost: 762
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Cost: 575
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest']
Cost: 607
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Rimnicu Vilcea', 'Sibiu',
'Fagaras', 'Bucharest']
Cost: 1119
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Bucharest']
Cost: 733
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Cost: 838
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu Vilcea', 'Sibiu', 'Fagaras',
'Bucharest']
Cost: 1030
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Craiova', 'Pitesti', 'Bucharest']
Cost: 605
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Cost: 418
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
Cost: 450
```

Breadth first Search:

Algorithm:

The algorithm uses queues to find all the possible paths from Arad to Bucharest.

Implementation in Python:

```
def bfs(graph1, start, goal):
    q = [(start,[start])]
    while q:
        (v, path)=q.pop(0)
        for n in graph1[v] - set(path):
            if n==goal:
                print(path+[n])
            else:
                q.append((n,path+[n]))

print("Breath First Search:")
bfs(graph, 'Arad', 'Bucharest');
```

Output:

['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Rimnicu Vilcea', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu Vilcea', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']

Comparing the costs:

Algorithm	Best	Worst
A-Star Algorithm	418	418
Depth First Search	418	1119
Breadth First Search	418	1119

Conclusion:

As it is obvious from the result that DFS and BFS can not guarantee the best path. It can either give us the best or can give us the path with longest cost as it depends on the the input and implementation. But A-Star is always going to return the Shortest Distance.