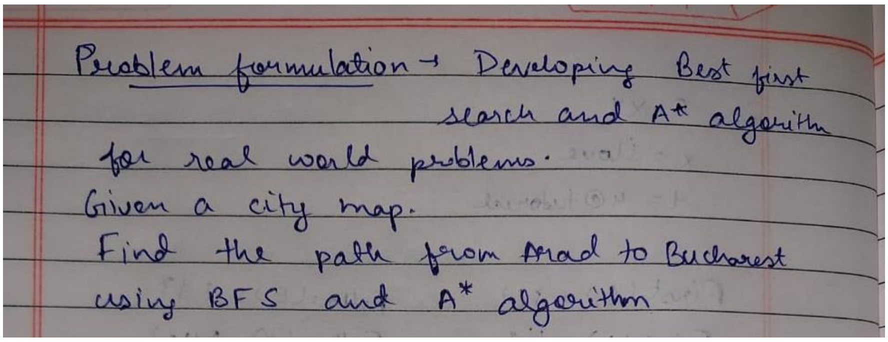
Prajwal Gupta

RA1911003010660

AI LAB 5

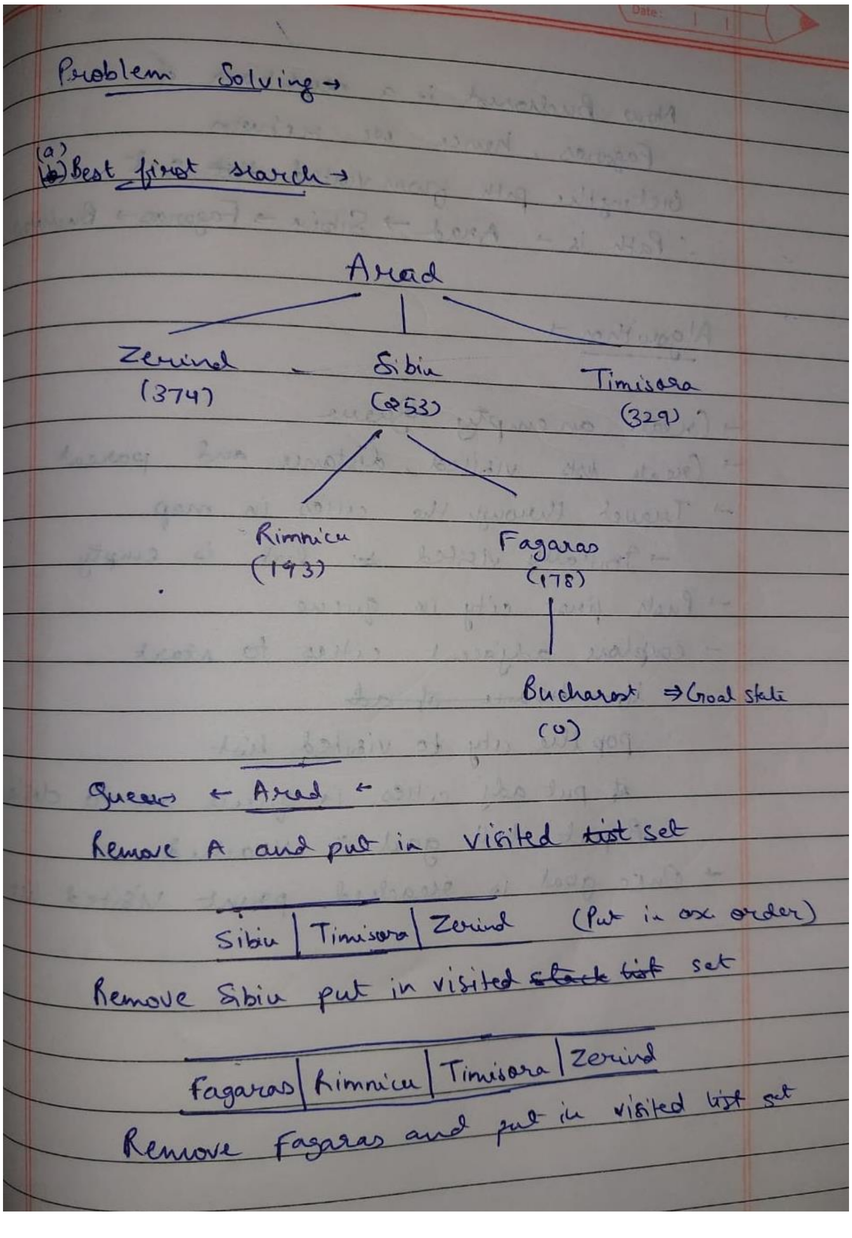
Aim- (A) Developing Best first search Algorithm for real world problems



Problem formulation → Developing Best first search and A* algorithm for real world problems.

Given a city map.

Find the path from Arad to Bucharest using BFS and A* algorithm



Problem Solving →

~~(b)~~ (a) Best first search →

Arad
Zerind
(374)
Sibiu
(253)
Timisora
(329)
Rimnicu
(193)
Fagaras
(178)
Bucharest ⇒ Goal state
(0)

Queue ← Arad ←

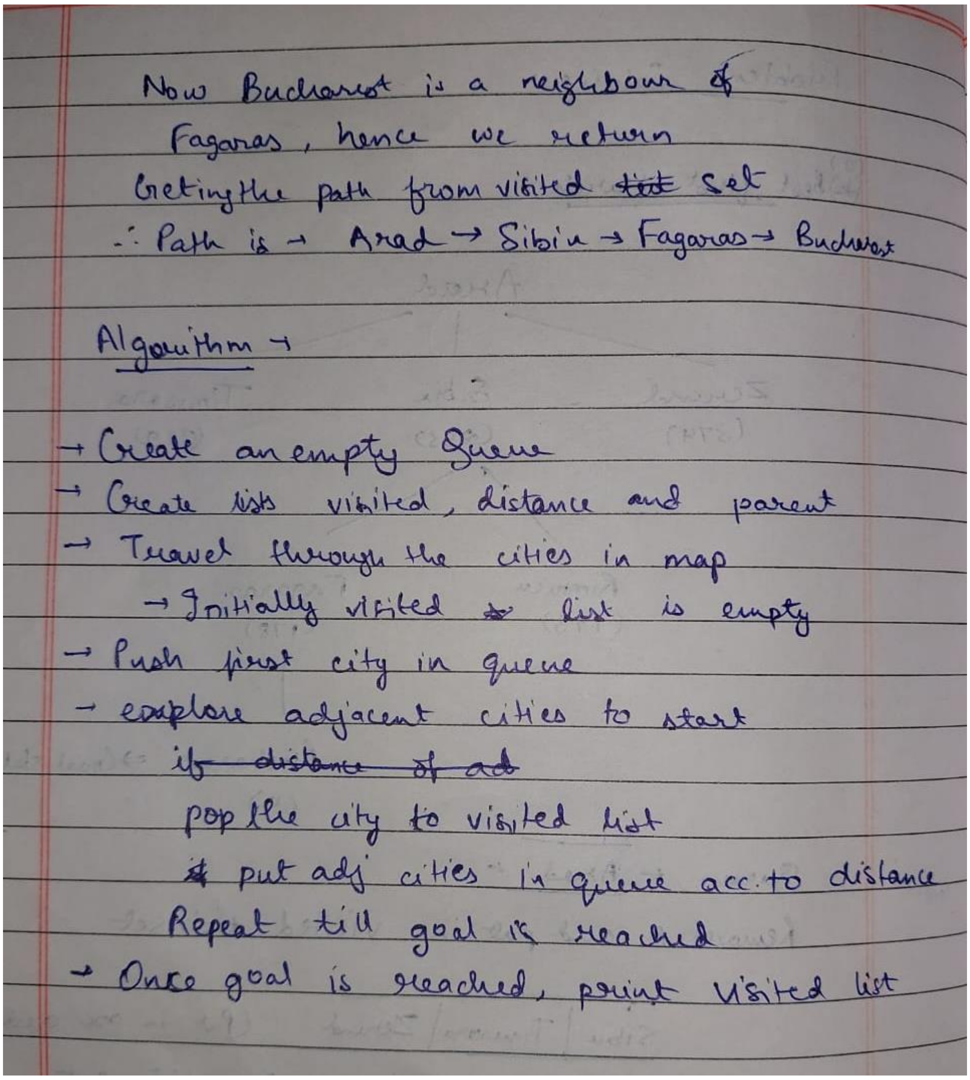
Remove A and put in visited ~~list~~ set

Sibiu | Timisora | Zerind (Put in asc order)

Remove Sibiu put in visited ~~stack~~ ~~list~~ set

Fagaras | Rimnicu | Timisora | Zerind

Remove Fagaras and put in visited list set



Now Buchareat is a neighbour of Fagaras, hence we return

Geting the path from visited ~~list~~ set

∴ Path is → Arad → Sibiu → Fagaras → Buchrest

Algorithm →

→ Create an empty Queue

→ Create lists visited, distance and parent

→ Travel through the cities in map

→ Initially visited ~~s~~ list is empty

→ Push first city in queue

→ explore adjacent cities to start

~~if distance of ad~~

pop the city to visited list

~~&~~ put adj cities in queue acc. to distance

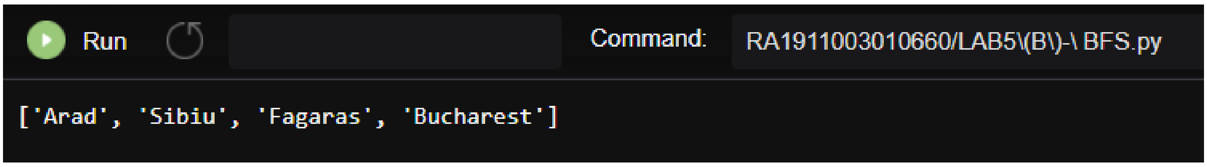
Repeat till goal is reached

→ Once goal is reached, print visited list

Code-

```
from queue import Queue

romaniaMap = {

    'Arad': ['Sibiu', 'Zerind', 'Timisoara'],

    'Zerind': ['Arad', 'Oradea'],

    'Oradea': ['Zerind', 'Sibiu'],

    'Sibiu': ['Arad', 'Oradea', 'Fagaras', 'Rimnicu'],

    'Timisoara': ['Arad', 'Lugoj'],

    'Lugoj': ['Timisoara', 'Mehadia'],

    'Mehadia': ['Lugoj', 'Drobeta'],

    'Drobeta': ['Mehadia', 'Craiova'],

    'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],

    'Rimnicu': ['Sibiu', 'Craiova', 'Pitesti'],
    'Fagaras': ['Sibiu', 'Bucharest'],
    'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],
    'Bucharest': ['Fagaras', 'Pitesti', 'Giurgiu', 'Urziceni'],
    'Giurgiu': ['Bucharest'],
    'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],
    'Hirsova': ['Urziceni', 'Eforie'],
    'Eforie': ['Hirsova'],
    'Vaslui': ['Iasi', 'Urziceni'],
    'Iasi': ['Vaslui', 'Neamt'],
    'Neamt': ['Iasi']
}

def bfs(startingNode, destinationNode):
    # For keeping track of what we have visited
    visited = {}
    # keep track of distance
    distance = {}
    # parent node of specific graph
    parent = {}

    bfs_traversal_output = []
    # BFS is queue based so using 'Queue' from python built-in
    queue = Queue()

    # travelling the cities in map
    for city in romaniaMap.keys():
        # since intially no city is visited so there will be nothing in visited list
        visited[city] = False
        parent[city] = None
        distance[city] = -1

    # starting from 'Arad'
    startingCity = startingNode
    visited[startingCity] = True
    distance[startingCity] = 0
    queue.put(startingCity)

    while not queue.empty():
        u = queue.get()     # first element of the queue, here it will be 'arad'
        bfs_traversal_output.append(u)

        # explore the adjust cities adj to 'arad'
        for v in romaniaMap[u]:
            if not visited[v]:
                visited[v] = True
                parent[v] = u
                distance[v] = distance[u] + 1
                queue.put(v)

        # reaching our destination city i.e 'bucharest'
    g = destinationNode
    path = []
    while g is not None:
        path.append(g)
        g = parent[g]

    path.reverse()
    # printing the path to our destination city
    print(path)
    # print(distance)
# Starting City & Destination City
bfs('Arad', 'Bucharest')
```

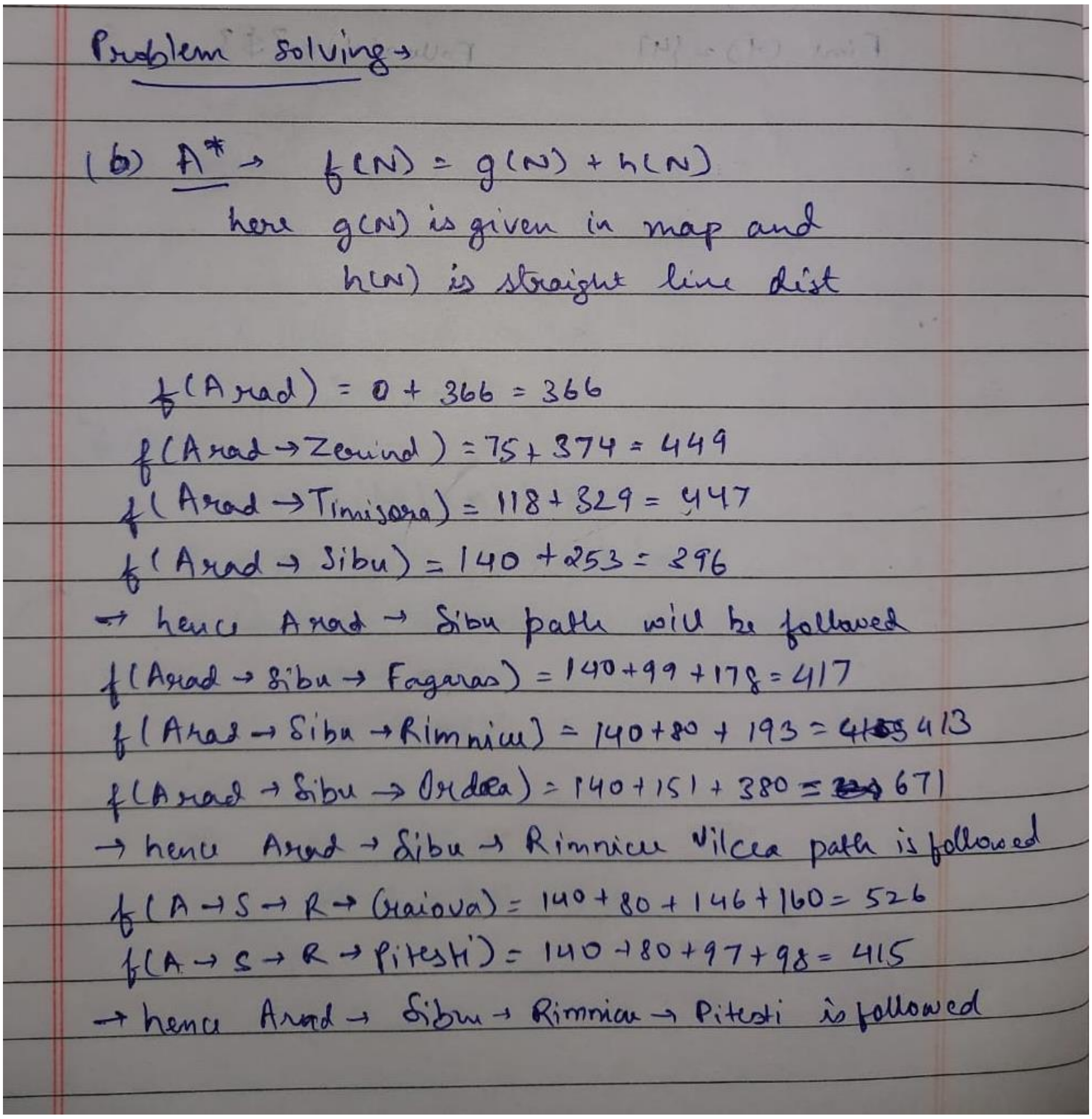
Output-



Command: RA1911003010660/LAB5\(B\)-\ BFS.py

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

Aim- (B) Developing A* Algorithm for real world problems



Problem solving →

(b) A* → $f(N) = g(N) + h(N)$

here g(N) is given in map and
h(N) is straight line dist

f(Arad) = 0 + 366 = 366

f(Arad → Zerind) = 75 + 374 = 449

f(Arad → Timisora) = 118 + 329 = 447

f(Arad → Sibu) = 140 + 253 = 396

→ hence Arad → Sibu path will be followed

f(Arad → Sibu → Fagaras) = 140 + 99 + 178 = 417

f(Arad → Sibu → Rimnicu) = 140 + 80 + 193 = ~~415~~ 413

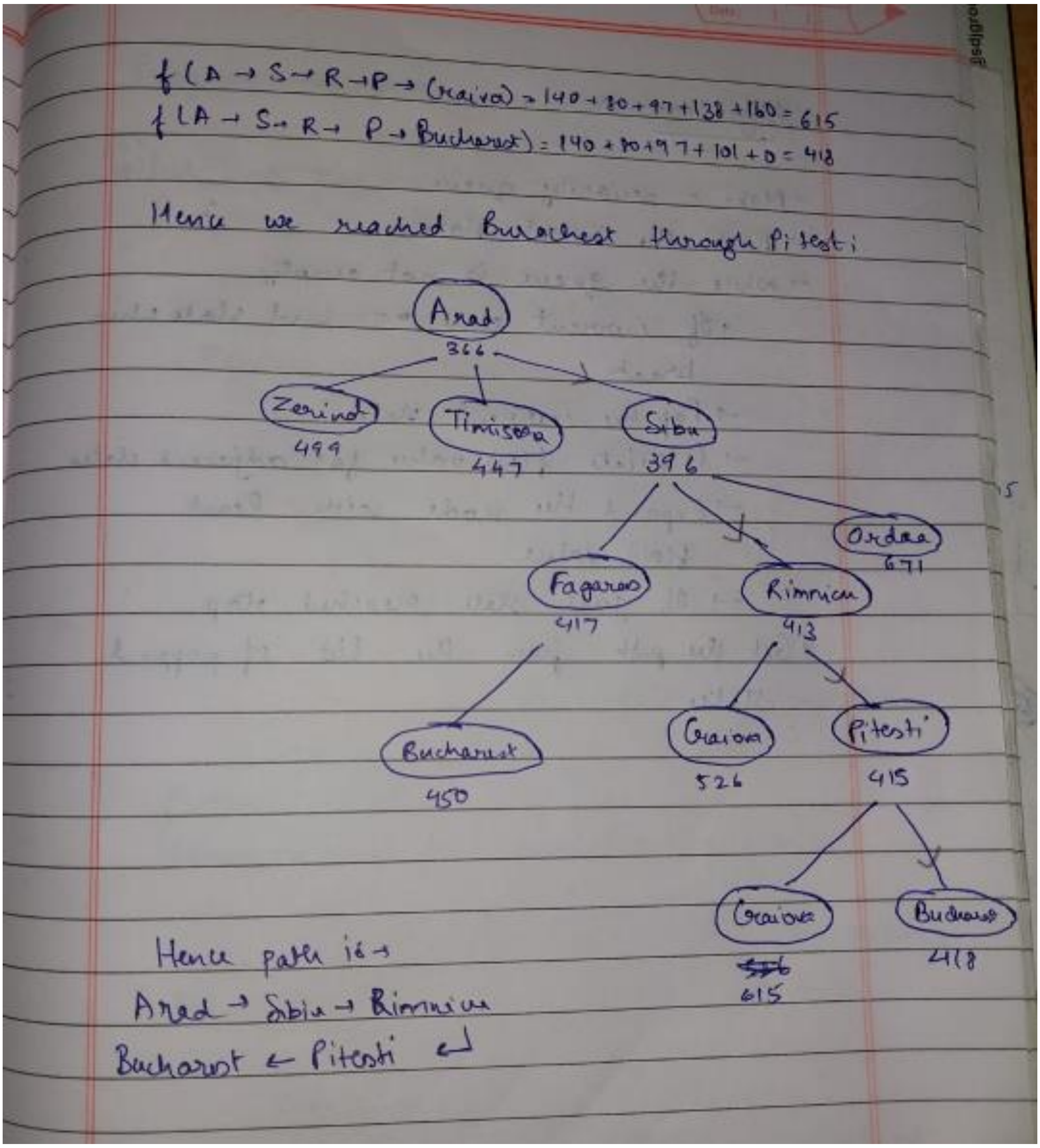
f(Arad → Sibu → Ordea) = 140 + 151 + 380 = 671

→ hence Arad → Sibu → Rimnicu Vilcea path is followed

f(A → S → R → Craiova) = 140 + 80 + 146 + 160 = 526

f(A → S → R → Pitesti) = 140 + 80 + 97 + 98 = 415

→ hence Arad → Sibu → Rimnicu → Pitesti is followed



f(A → S → R → P → Craiva) = 140 + 80 + 97 + 138 + 160 = 615

f(A → S → R → P → Bucharest) = 140 + 80 + 97 + 101 + 0 = 418

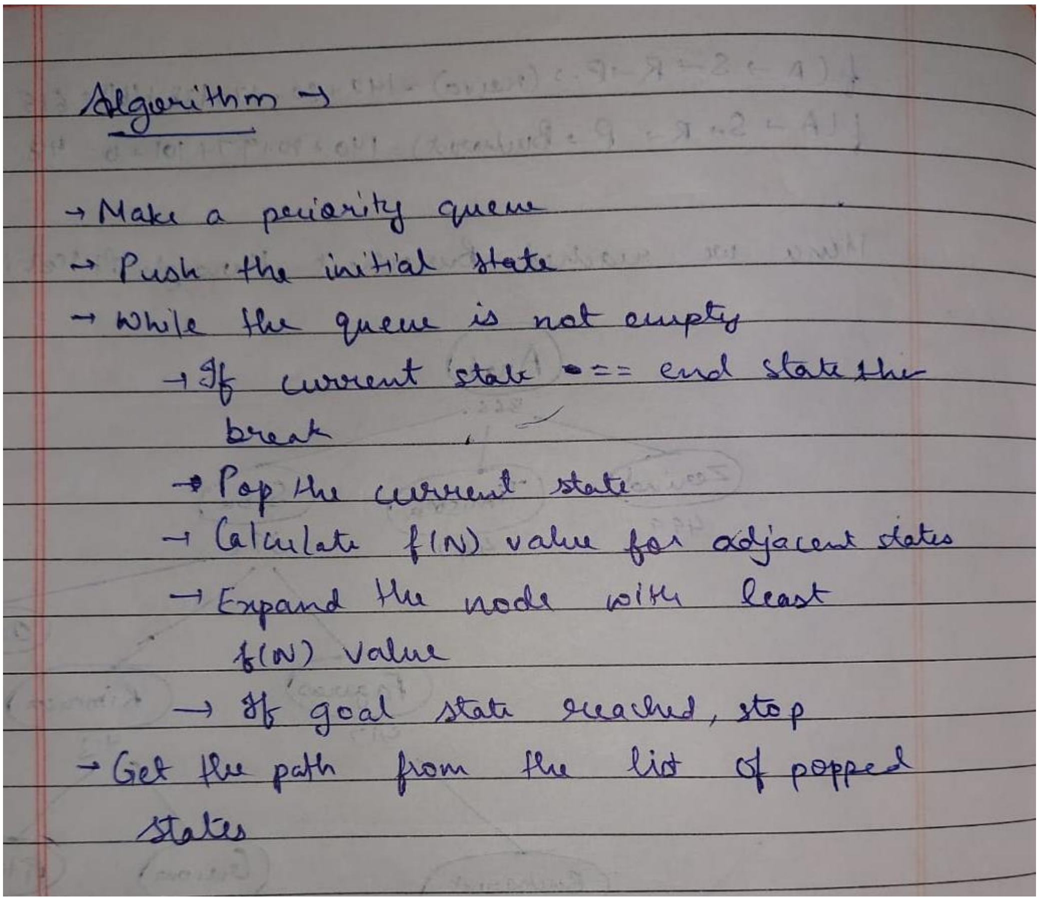
Hence we reached Buracheat through Pitest;

Arad
366
Zerind
499
Timisoara
447
Sibu
396
Fagaras
417
Rimnicu
413
Ordaa
671
Bucharest
450
Craiova
526
Pitesti
415
Craiova
~~526~~
615
Bucharest
418

Hence path is →

Arad → Sbiu → Rimnicu

Bucharest ← Pitesti ↵



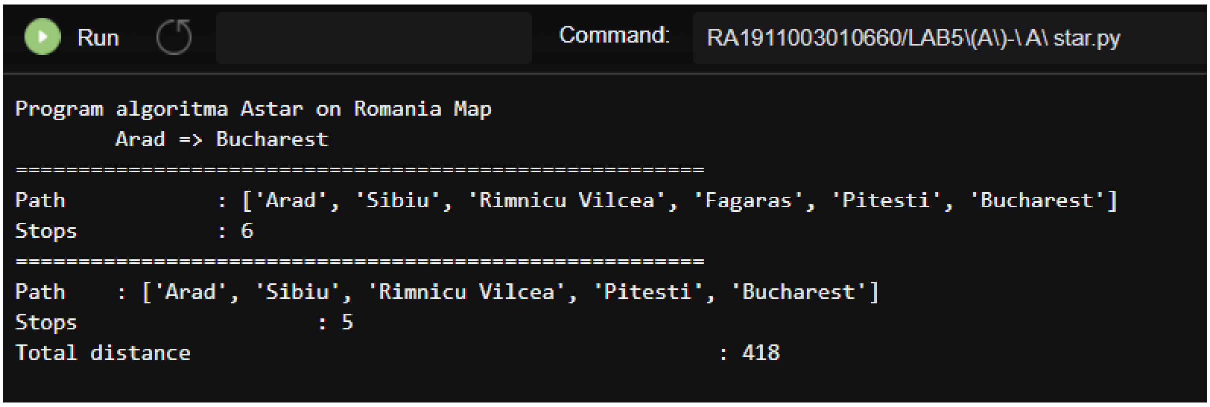
Algorithm →

```
→ Make a periority queue
→ Push the initial state
→ While the queue is not empty
    → If current state == end state the
      break
    → Pop the current state
    → Calculate f(N) value for adjacent states
    → Expand the node with least
      f(N) value
    → If goal state reached, stop
→ Get the path from the list of popped
  states
```

Code-

```
import heapq



class priorityQueue:
    def __init__(self):
        self.cities = []

    def push(self, city, cost):
        heapq.heappush(self.cities, (cost, city))

    def pop(self):
        return heapq.heappop(self.cities)[1]

    def isEmpty(self):
        if (self.cities == []):
            return True
        else:
            return False

    def check(self):
        print(self.cities)



class ctNode:
    def __init__(self, city, distance):
        self.city = str(city)
        self.distance = str(distance)



romania = {}



def makedict():
    file = open("RA1911003010660/LAB5-a.txt", 'r')
    for string in file:
        line = string.split(',')
        ct1 = line[0]
        ct2 = line[1]
        dist = int(line[2])
        romania.setdefault(ct1, []).append(ctNode(ct2, dist))
        romania.setdefault(ct2, []).append(ctNode(ct1, dist))



def makehuristikdict():
    h = {}
    with open("RA1911003010660/LAB5-b.txt", 'r') as file:
        for line in file:
            line = line.strip().split(",")
            node = line[0].strip()
            sld = int(line[1].strip())
            h[node] = sld
    return h


def heuristic(node, values):
    return values[node]


def astar(start, end):
    path = {}
    distance = {}
    q = priorityQueue()
    h = makehuristikdict()

    q.push(start, 0)
    distance[start] = 0
    path[start] = None
    expandedList = []

    while (q.isEmpty() == False):
        current = q.pop()
        expandedList.append(current)

        if (current == end):
            break

        for new in romania[current]:
            g_cost = distance[current] + int(new.distance)

            # print(new.city, new.distance, "now : " + str(distance[current]), g_cost)

            if (new.city not in distance or g_cost < distance[new.city]):
                distance[new.city] = g_cost
                f_cost = g_cost + heuristic(new.city, h)
                q.push(new.city, f_cost)
                path[new.city] = current

    printoutput(start, end, path, distance, expandedList)

def printoutput(start, end, path, distance, expandedlist):
    finalpath = []
    i = end

    while (path.get(i) != None):
        finalpath.append(i)
        i = path[i]
    finalpath.append(start)
    finalpath.reverse()
    print("Program algoritma Astar on Romania Map")
    print("\tArad => Bucharest")
    print("============================================================")
    print("Path \t\t: " + str(expandedlist))
    print("Stops \t\t: " + str(len(expandedlist)))
    print("============================================================")
    print("Path\t: " + str(finalpath))
    print("Stops \t\t\t: " + str(len(finalpath)))
    print("Total distance \t\t\t\t\t\t: " + str(distance[end]))

def main():
    src = "Arad"
    dst = "Bucharest"
    makedict()
    astar(src, dst)
if __name__ == "__main__":
    main()
```

Output-



Command: RA1911003010660/LAB5\(A\)-\ A\ star.py

```
Program algoritma Astar on Romania Map
        Arad => Bucharest
=======================================================
Path            : ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Fagaras', 'Pitesti', 'Bucharest']
Stops           : 6
=======================================================
Path    : ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Stops                   : 5
Total distance                                          : 418
```