heuristic\_value = {}

map = {}

with open("D://CSE BRACU//CSE422//Lab//Lab1//Input file.txt", "r") as input\_file:

    input\_list = input\_file.readlines()

    for line in input\_list:

        parent = line.split()

        heuristic\_value[parent[0]] = int(parent[1])

        map[parent[0]] = []

        count = 2

        for i in range(len(parent[2:])//2):

            map[parent[0]].append((int(parent[count+1]), parent[count]))

            count+=2

print(map)

print(heuristic\_value)

import heapq

def astar\_search\_path\_finding(start, goal, graph, heuristic):

    pri\_queue = []

    f\_start = heuristic[start] + 0

    heapq.heappush(pri\_queue, (f\_start, start))

    g\_value = {start:0}

    parent = {start:None}

    found  = False

    while len(pri\_queue) > 0:

        current\_node = heapq.heappop(pri\_queue)      # current\_node = (cost, city)

        if current\_node[1] == goal:

            found = True

            break

        for child in graph[current\_node[1]]:

            g\_new = g\_value[current\_node[1]] + child[0]

            if child[1] not in g\_value or g\_new < g\_value[child[1]]:

                g\_value[child[1]] = g\_new

                f\_new = g\_new + heuristic[child[1]]

                heapq.heappush(pri\_queue, (f\_new, child[1]))

                parent[child[1]] = current\_node[1]

    if found == False:

        print('NO PATH FOUND')

    else:

        path = [goal]

        node = goal

        while node != start:

            node = parent[node]

            path.append(node)

        idx = len(path)-1

        path\_str = ''

        while idx != -1:

            if idx == 0:

                path\_str += path[idx]

            else:

                path\_str += path[idx]+' --> '

            idx -= 1

        print(f'Path: {path\_str}')

        print(f'Total distance: {g\_value[goal]} km')

starting\_node = 'Arad'

goal\_node = 'Bucharest'

astar\_search\_path\_finding(starting\_node, goal\_node, map, heuristic\_value)