class GraphAndHeuristicBuilder:

def \_\_init\_\_(self, file\_path):

self.graph = {}

self.heuristic = {}

self.file\_path = file\_path

def create\_graph\_and\_heuristic(self):

file = open(self.file\_path, "r")

for line in file:

line\_arr = line.split()

self.populate\_graph\_and\_heuristic(line\_arr)

def populate\_graph\_and\_heuristic(self, line\_arr):

for i in range(len(line\_arr)):

if i == 0:

self.graph[line\_arr[i]] = []

elif i == 1:

self.heuristic[line\_arr[0]] = int(line\_arr[i])

else:

if i % 2 == 0:

self.graph[line\_arr[0]].append((line\_arr[i], int(line\_arr[i + 1])))

def print(self):

print("Graph:")

for key, value in self.graph.items():

print(f'{key}: {value}')

print("--------------------------------")

print("Heuristic Table:")

for key, value in self.heuristic.items():

print(f'{key}: {value}')

print("---------------------------------")

def get\_graph\_heuristic(self):

return self.graph, self.heuristic

class A\_star:

def \_\_init\_\_(self, graph, heuristic, start=None, goal=None):

self.graph = graph

self.heuristic = heuristic

self.start = start

self.goal = goal

def set\_start\_and\_goal(self):

start = input("Start: ")

goal = input("Destination: ")

self.start = start

self.goal = goal

def check\_variables(self):

if self.graph == None or self.heuristic == None:

print("Please create graph and heuristic using the GraphAndHeuristicBuilder class")

return False

elif self.start == None or self.goal == None:

print("Please set start and goal using parameters or set\_start\_and\_goal method")

return False

else:

return True

def run\_a\_star(self):

run = self.check\_variables()

if run == False:

return

priority\_queue = []

priority\_queue.append((self.start, 0))

came\_from = []

goal\_cost = 99999

goal\_counter = 0

while True:

node = priority\_queue.pop(0)[0]

goal = False

if node == self.goal:

break

for adj\_node, cost in self.graph[node]:

total\_cost = 0

node\_cost = 0

if adj\_node in came\_from:

continue

else:

temp\_node = node

for i in range(len(came\_from) - 1, -1, -1):

for tupl in self.graph[temp\_node]:

if tupl[0] == came\_from[i]:

node\_cost += tupl[1]

temp\_node = tupl[0]

break

total\_cost = cost + node\_cost + self.heuristic[adj\_node]

if adj\_node == self.goal and total\_cost < goal\_cost:

goal\_cost = goal\_cost

goal = True

if goal\_counter == 0:

came\_from.append(node)

goal\_counter += 1

else:

came\_from[-1] = node

priority\_queue.append((adj\_node, total\_cost))

if goal == False:

came\_from.append(node)

priority\_queue.sort(key=lambda x: x[1])

self.print\_path\_distance(came\_from)

def print\_path\_distance(self, came\_from):

distance = 0

temp\_node = self.goal

path = "Bucharest"

for i in range(len(came\_from) - 1, -1, -1):

for tupl in self.graph[temp\_node]:

if tupl[0] == came\_from[i]:

distance += tupl[1]

temp\_node = tupl[0]

break

path = temp\_node + " -> " + path

print("Path:", path)

print("Distance:", distance)

graph\_heuristic = GraphAndHeuristicBuilder("Lab - 1/Input file.txt")

graph\_heuristic.create\_graph\_and\_heuristic()

graph\_heuristic.print()

graph, heuristic = graph\_heuristic.get\_graph\_heuristic()

a\_star = A\_star(graph, heuristic)

a\_star.set\_start\_and\_goal()

a\_star.run\_a\_star()