**CSE422\_01\_Lab\_Assignment01\_Turnitin\_Summer2024**

**Name : Md. Anwar Hossain | Student ID : 22101710**

import heapq

def A\_star\_Search(graph, heuristics, start, end):

nodes = [(0, start)]

path = {}

values = {position: float('inf') for position in graph}

values[start] = 0

while nodes:

\_, current\_node = heapq.heappop(nodes)

if current\_node == end:

path\_sequence = restore\_track(path, current\_node)

return path\_sequence

for adjacent\_city, cost in graph[current\_node]:

probable\_value = values[current\_node] + cost

if probable\_value < values[adjacent\_city]:

path[adjacent\_city] = current\_node

values[adjacent\_city] = probable\_value

total\_value = probable\_value + heuristics[adjacent\_city]

heapq.heappush(nodes, (total\_value, adjacent\_city))

return None

def restore\_track(path, current\_node):

path\_sequence = [current\_node]

while current\_node in path:

current\_node = path[current\_node]

path\_sequence.insert(0, current\_node)

return path\_sequence

# Open files

with open('22101710\_Md.\_Anwar\_Hossain\_CSE422\_03\_Lab\_Assignment01\_InputFile\_Summer2024.txt', 'r') as f1, open('22101710\_Md.\_Anwar\_Hossain\_CSE422\_03\_Lab\_Assignment01\_OutputFile\_Summer2024.txt', 'w') as f2:

all\_lines = f1.readlines()

graph = {}

heuristics = {}

for x in all\_lines:

pieces = x.split()

position = pieces[0]

heuristic = int(pieces[1])

next\_to = []

for i in range(2, len(pieces), 2):

city = pieces[i]

distance = int(pieces[i + 1])

next\_to.append((city, distance))

graph[position] = next\_to

heuristics[position] = heuristic

# print(graph)

# print('----------------------------------------------------------------')

# print(heuristics)

start\_position = input('Start node: ')

end\_position = input('Destination: ')

path\_sequence = A\_star\_Search(graph, heuristics, start\_position, end\_position)

if path\_sequence:

total\_distance = 0

for i in range(len(path\_sequence) - 1):

current\_position = path\_sequence[i]

next\_position = path\_sequence[i + 1]

for adjacent\_position, cost in graph[current\_position]:

if adjacent\_position == next\_position:

total\_distance += cost

break

f2.write("Path: " + " -> ".join(path\_sequence) + "\n")

f2.write(f"Total distance: {total\_distance} km\n")

else:

f2.write(f"No path found from {start\_position} to {end\_position}\n")