import heapq

def read\_graph(file\_path):

graph = {}

heuristics = {}

with open(file\_path, 'r') as f:

for line in f:

parts = line.split()

node = parts[0]

heuristics[node] = int(parts[1])

neighbors = []

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

neighbor = parts[i]

distance = int(parts[i + 1])

neighbors.append((neighbor, distance))

graph[node] = neighbors

return graph, heuristics

def a\_star\_search(graph, heuristics, start, goal):

open\_set = []

heapq.heappush(open\_set, (heuristics[start], start))

came\_from = {}

g\_score = {node: float('inf') for node in graph}

g\_score[start] = 0

f\_score = {node: float('inf') for node in graph}

f\_score[start] = heuristics[start]

while open\_set:

\_, current = heapq.heappop(open\_set)

if current == goal:

total\_path = [current]

while current in came\_from:

current = came\_from[current]

total\_path.append(current)

total\_path.reverse()

return total\_path, g\_score[goal]

for neighbor, distance in graph[current]:

tentative\_g\_score = g\_score[current] + distance

if tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = g\_score[neighbor] + heuristics[neighbor]

if neighbor not in [i[1] for i in open\_set]:

heapq.heappush(open\_set, (f\_score[neighbor], neighbor))

return None, float('inf')

def main():

input\_file\_path = 'input.txt'

output\_file\_path = 'output.txt'

graph, heuristics = read\_graph(input\_file\_path)

start = input("Start node: ").strip()

goal = input("Destination: ").strip()

path, total\_distance = a\_star\_search(graph, heuristics, start, goal)

with open(output\_file\_path, 'w') as output\_file:

if path:

output\_file.write(f"Path: {' -> '.join(path)}\n")

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

else:

output\_file.write("NO PATH FOUND\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()