import heapq

snode = input("Start node: ")

gnode = input("Destination: ")

with open('input\_file.txt', 'r') as file: #read text file and generate dictionary to use in the algorithm

ndict = {}

hval = {}

while True:

line = file.readline()

if not line:

break

temp = line.split()

sval, hnum = temp[0], int(temp[1])

i = 2

nnode = {}

while i < len(temp):

ntemp, dtemp = temp[i], int(temp[i + 1])

nnode[ntemp] = dtemp

i += 2

ndict[sval], hval[sval] = nnode, hnum

def astar(ndict, s, g, heur): #a star algorithm function

dist = {i: float('inf') for i in ndict}

dist[s] = 0

pnode = {i: 0 for i in ndict}

edist = {i: float('inf') for i in ndict}

edist[s] = heur(s, g)

pqueue = [(edist[s], s)]

heapq.heapify(pqueue)

while pqueue:

cdist, cnode = heapq.heappop(pqueue)

if cnode == g:

return pcalc(pnode, g)

for nnode, distance in ndict[cnode].items():

new\_distance = dist[cnode] + distance

if new\_distance < dist[nnode]:

dist[nnode], edist[nnode] = new\_distance, new\_distance + heur(nnode, g)

pnode[nnode] = cnode

heapq.heappush(pqueue, (edist[nnode], nnode))

return None

def heur(x, y): #Returns the heuristic value of a node

return hval[y]

def pcalc(pnode, g): # Returns the shortest path from a goal node to the start node

path = []

node = g

while node:

path.append(node)

node = pnode[node]

return path[::-1]

def pdist(ndict, path): #Returns the optimal path with name

dist = 0

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

dist += ndict[path[i]][path[i+1]]

return dist

def find(): #Main function that starts other functions and returns output

path = astar(ndict, snode, gnode, heur)

if path:

distance = pdist(ndict, path)

print("Path:", " -> ".join(path))

print("Total Distance:", distance, "km")

else:

print("NO PATH FOUND")

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

find()