A\* search

import heapq

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

queue = [(heuristics[start], 0, start, [start])] # (f = g + h, g, node, path)

visited = set()

while queue:

f, g, current, path = heapq.heappop(queue)

if current == goal:

print(f"\nReached goal {goal} with total cost {g}")

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

return

if current not in visited:

visited.add(current)

for neighbor, cost in graph.get(current, []):

if neighbor not in visited:

g\_new = g + cost

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

heapq.heappush(queue, (f\_new, g\_new, neighbor, path + [neighbor]))

print("Goal not reachable.")

# ---------- USER INPUT ----------

graph = {}

heuristics = {}

n = int(input("Enter number of nodes: "))

# Input heuristic values

print("\nEnter heuristic values:")

for \_ in range(n):

node = input("Node name: ")

h = int(input(f"Heuristic value of {node}: "))

heuristics[node] = h

# Input weighted graph (adjacency list)

print("\nEnter neighbors and edge costs:")

for \_ in range(n):

node = input("Enter node: ")

neighbor\_input = input(f"Enter neighbors and costs for {node} (format: B 2 C 5): ").split()

neighbors = []

for i in range(0, len(neighbor\_input), 2):

neighbor = neighbor\_input[i]

cost = int(neighbor\_input[i+1])

neighbors.append((neighbor, cost))

graph[node] = neighbors

start\_node = input("\nEnter start node: ")

goal\_node = input("Enter goal node: ")

# ---------- RUN A\* SEARCH ----------

print("\nA\* Search Trace:")

a\_star\_search(graph, heuristics, start\_node, goal\_node)

OUTPUT:

# ---------- SAMPLE INPUT ----------

# Enter number of nodes: 6

#

# Enter heuristic values:

# Node name: A

# Heuristic value of A: 6

# Node name: B

# Heuristic value of B: 4

# Node name: C

# Heuristic value of C: 5

# Node name: D

# Heuristic value of D: 2

# Node name: E

# Heuristic value of E: 1

# Node name: F

# Heuristic value of F: 0

#

# Enter neighbors and edge costs:

# Enter node: A

# Enter neighbors and costs for A (format: B 2 C 4): B 1 C 4

# Enter node: B

# Enter neighbors and costs for B (format: D 2 E 5): D 2 E 5

# Enter node: C

# Enter neighbors and costs for C (format: E 1): E 1

# Enter node: D

# Enter neighbors and costs for D (format: F 3): F 3

# Enter node: E

# Enter neighbors and costs for E (format: F 1): F 1

# Enter node: F

# Enter neighbors and costs for F (format: ):

#

# Enter start node: A

# Enter goal node: F

#

# ---------- OUTPUT ----------

# A\* Search Trace:

# Reached goal F with total cost 5

# Path: A -> B -> D -> F