import heapq

graph = {

's': {'b': 4, 'c': 3},

'b': {'s': 4, 'f': 5, 'e': 12},

'f': {'b': 5, 'g': 16},

'c': {'s': 3, 'e': 10, 'd': 7},

'd': {'c': 7, 'e': 2},

'e': {'b': 12, 'c': 10, 'd': 2, 'g': 5},

'g': {'f': 16, 'e': 5}

}

heuristics = {

's': 14,

'b': 12,

'f': 11,

'c': 11,

'd': 6,

'e': 4,

'g': 0

}

def astar(start, goal):

open\_list, closed\_list, parent = [(heuristics[start], 0, start)], set(), {}

while open\_list:

\_, cost, current = heapq.heappop(open\_list)

if current == goal:

path = [current]

while current in parent:

path.append(current := parent[current])

return path[::-1]

closed\_list.add(current)

for neighbor, edge\_cost in graph[current].items():

if neighbor not in closed\_list:

new\_cost = cost + edge\_cost

heapq.heappush(open\_list, (new\_cost + heuristics[neighbor], new\_cost, neighbor))

parent[neighbor] = current

return None

start\_node = 's'

goal\_node = 'g'

path = astar(start\_node, goal\_node)

if path:

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

else:

print("No path found.")