import heapq

# Define a simple graph with distances and emissions (example data)

graph = {

'A': {'B': (5, 10), 'C': (2, 4)},

'B': {'A': (5, 10), 'C': (8, 16), 'D': (7, 14)},

'C': {'A': (2, 4), 'B': (8, 16), 'D': (3, 6)},

'D': {'B': (7, 14), 'C': (3, 6)}

}

def dijkstra(graph, start, end, weight\_index=0):

"""

Dijkstra's algorithm to find the shortest or lowest-emission path.

weight\_index: 0 for distance, 1 for emission.

"""

queue = [(0, start, [])]

seen = set()

while queue:

(cost, node, path) = heapq.heappop(queue)

if node in seen:

continue

seen.add(node)

path = path + [node]

if node == end:

return (cost, path)

for next\_node, (distance, emission) in graph.get(node, {}).items():

weight = (distance, emission)[weight\_index]

heapq.heappush(queue, (cost + weight, next\_node, path))

return (float('inf'), [])

# Example runs

start = 'A'

end = 'D'

# Optimize for shortest distance

distance\_cost, distance\_path = dijkstra(graph, start, end, weight\_index=0)

print(f"Shortest distance path from {start} to {end}: {distance\_path} (Total distance: {distance\_cost})")

# Optimize for lowest CO₂ emission

emission\_cost, emission\_path = dijkstra(graph, start, end, weight\_index=1)

print(f"Lowest emission path from {start} to {end}: {emission\_path} (Total emissions: {emission\_cost})")