import heapq

def is\_goal(jug4, jug3):

return jug4 == 2

def next\_states(jug4, jug3, path):

states = []

# Fill jug4

states.append((4, jug3, path + ["Fill 4-gallon jug"]))

# Fill jug3

states.append((jug4, 3, path + ["Fill 3-gallon jug"]))

# Pour from jug4 to jug3

pour\_amount = min(jug4, 3 - jug3)

states.append((jug4 - pour\_amount, jug3 + pour\_amount, path + ["Pour from 4-gallon to 3-gallon jug"]))

# Pour from jug3 to jug4

pour\_amount = min(jug3, 4 - jug4)

states.append((jug4 + pour\_amount, jug3 - pour\_amount, path + ["Pour from 3-gallon to 4-gallon jug"]))

# Empty jug4

states.append((0, jug3, path + ["Empty 4-gallon jug"]))

# Empty jug3

states.append((jug4, 0, path + ["Empty 3-gallon jug"]))

return states

def heuristic(state):

jug4, jug3 = state

# Heuristic estimate: Absolute difference between jug4 and the goal amount

return abs(jug4 - 2)

def astar\_search(start\_jug4, start\_jug3):

heap = [(heuristic((start\_jug4, start\_jug3)), (start\_jug4, start\_jug3, []))]

visited = set()

while heap:

h\_score, (jug4, jug3, path) = heapq.heappop(heap)

if (jug4, jug3) in visited:

continue

visited.add((jug4, jug3))

if is\_goal(jug4, jug3):

return path

for next\_state in next\_states(jug4, jug3, path):

g\_score = len(next\_state[2]) # Actual cost from start state to current state

f\_score = g\_score + heuristic(next\_state[:2]) # f-score = g-score + heuristic

heapq.heappush(heap, (f\_score, next\_state))

return None

start\_jug4 = 0

start\_jug3 = 0

solution = astar\_search(start\_jug4, start\_jug3)

if solution:

print("Steps to get precisely 2 gallons of water in a 4-gallon jug:")

for step in solution:

print(step)

else:

print("Solution not found!")