**PROGRAM:**

from heapq import heappop, heappush

def heuristic(state, target, target\_jug):

if target\_jug == 1:

return abs(state[0] - target)

else:

return abs(state[1] - target)

def a\_star(Jug1, Jug2, target, target\_jug):

open\_set = []

heappush(open\_set, (0 + heuristic((0, 0), target, target\_jug), 0, (0, 0), []))

closed\_set = set()

g\_cost = { (0, 0): 0 }

while open\_set:

\_, cost, current, path = heappop(open\_set)

if current in closed\_set:

continue

path = path + [current]

closed\_set.add(current)

if (target\_jug == 1 and current[0] == target) or (target\_jug == 2 and current[1] == target):

for state in path:

print(f"({state[0]}, {state[1]})")

return

for next\_state in get\_neighbors(current, Jug1, Jug2):

if next\_state in closed\_set:

continue

tentative\_g\_cost = cost + 1

if next\_state not in g\_cost or tentative\_g\_cost < g\_cost[next\_state]:

g\_cost[next\_state] = tentative\_g\_cost

f\_cost = tentative\_g\_cost + heuristic(next\_state, target, target\_jug)

heappush(open\_set, (f\_cost, tentative\_g\_cost, next\_state, path))

print("No solution")

def get\_neighbors(state, Jug1, Jug2):

neighbors = []

a, b = state

neighbors.append((Jug1, b))

neighbors.append((a, Jug2))

neighbors.append((0, b))

neighbors.append((a, 0))

transfer = min(a, Jug2 - b)

neighbors.append((a - transfer, b + transfer))

transfer = min(b, Jug1 - a)

neighbors.append((a + transfer, b - transfer))

return neighbors

Jug1 = int(input("Enter the capacity of Jug1: "))

Jug2 = int(input("Enter the capacity of Jug2: "))

target = int(input("Enter the target amount: "))

target\_jug = int(input("Enter the target jug (1 for Jug1, 2 for Jug2): "))

print("Path from initial state to solution state:")

a\_star(Jug1, Jug2, target, target\_jug)

**OUTPUT:**

