**Program:**

Breadth First Search:

from collections import deque

def printPath(target, parent):

res = []

curr = target

src = (-1, -1)

while(curr != src):

res.append(curr)

curr = parent[curr]

print()

while(len(res) != 0):

print(res.pop(), end = " -> ")

print("GOAL")

# Available operations:

# 1. Fill the jug

# 2. Empty the jug

# 3. Transfer jug contents

def water\_jug\_bfs(j1, j2, water):

visited = set() # To hold the already visited nodes

q = deque() # To hold the bfs queue

parent = dict() # To store parent of any node

q.append((0, 0)) # initially we start with (0, 0) as the starting state

visited.add((0, 0))

parent[(0, 0)] = (-1, -1) # the starting state has no parent

isSolvable = False # Sometimes problem cant be solved

target = [(0, water), (water, 0)] # required target state

while(len(q) != 0):

curr = q.popleft();

if(curr in target):

isSolvable = True

break

curr\_j1, curr\_j2 = curr[0], curr[1]

possiblities = []

possiblities.append((j1, curr\_j2)) # 1a) Fill jug1

possiblities.append((curr\_j1, j2)) # 1b) Fill jug2

possiblities.append((0, curr\_j2)) # 2a) Empty jug1

possiblities.append((curr\_j1, 0)) # 2b) Empty jug2

# 3a) Jug-1 to Jug-2

# cant transfer when jug-1 is empty and jug-2 is already full

if(curr\_j1 != 0 and curr\_j2 != j2):

total\_water = curr\_j1 + curr\_j2

# when total capacity is less than jug-2 capacity

if(total\_water <= j2): possiblities.append((0, total\_water))

# when total capacity is greater than jug-2 capacity

else: possiblities.append((total\_water-j2, j2))

# 3b) Jug-2 to Jug-1

# cant transfer when jug-2 is empty and jug-1 is already full

if(curr\_j1 != j1 and curr\_j2 != 0):

total\_water = curr\_j1 + curr\_j2

# when total capacity is less than jug-1 capacity

if(total\_water <= j1): possiblities.append((total\_water, 0))

# when total capacity is greater than jug-1 capacity

else: possiblities.append((j1, total\_water-j1))

for poss in possiblities:

if(poss not in visited):

x, y = poss[0], poss[1]

q.append((x, y))

visited.add((x, y))

parent[(x, y)] = curr

if(isSolvable):

printPath(curr, parent)

else:

print("Not possible to work with these inputs")

if \_\_name\_\_ == "\_\_main\_\_":

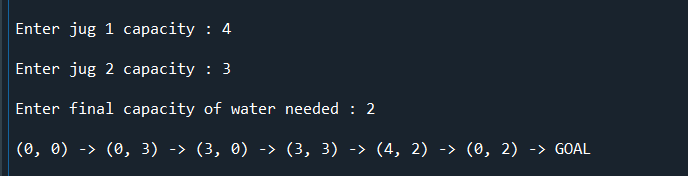
jug1 = int(input("Enter jug 1 capacity : "))

jug2 = int(input("Enter jug 2 capacity : "))

water = int(input("Enter final capacity of water needed : "))

water\_jug\_bfs(jug1, jug2, water)

**Output:**

****

Using State\_Space (BFS):

from collections import deque

def water\_jug\_bfs(j1, j2, water):

visited = set() # To hold the already visited nodes

q = deque() # To hold the bfs queue

q.append((0, 0)) # initially we start with (0, 0) as the starting state

print("\n", (0, 0))

visited.add((0, 0))

isSolvable = False # Sometimes problem cant be solved

target = [(0, water), (water, 0)] # required target state

while(len(q) != 0):

size = len(q)

print("\n\n \*\*\* \n")

for \_ in range(size):

curr = q.popleft();

if(curr in target):

isSolvable = True

break

curr\_j1, curr\_j2 = curr[0], curr[1]

possiblities = []

possiblities.append((j1, curr\_j2)) # 1a) Fill jug1

possiblities.append((curr\_j1, j2)) # 1b) Fill jug2

possiblities.append((0, curr\_j2)) # 2a) Empty jug1

possiblities.append((curr\_j1, 0)) # 2b) Empty jug2

# 3a) Jug-1 to Jug-2

# cant transfer when jug-1 is empty and jug-2 is already full

if(curr\_j1 != 0 and curr\_j2 != j2):

total\_water = curr\_j1 + curr\_j2

# when total capacity is less than jug-2 capacity

if(total\_water <= j2): possiblities.append((0, total\_water))

# when total capacity is greater than jug-2 capacity

else: possiblities.append((total\_water-j2, j2))

# 3b) Jug-2 to Jug-1

# cant transfer when jug-2 is empty and jug-1 is already full

if(curr\_j1 != j1 and curr\_j2 != 0):

total\_water = curr\_j1 + curr\_j2

# when total capacity is less than jug-1 capacity

if(total\_water <= j1): possiblities.append((total\_water, 0))

# when total capacity is greater than jug-1 capacity

else: possiblities.append((j1, total\_water-j1))

for poss in possiblities:

if(poss not in visited):

x, y = poss[0], poss[1]

q.append((x, y))

print((x, y), end= " ")

visited.add((x, y))

if(isSolvable == False):

print("Not possible to work with these inputs")

if \_\_name\_\_ == "\_\_main\_\_":

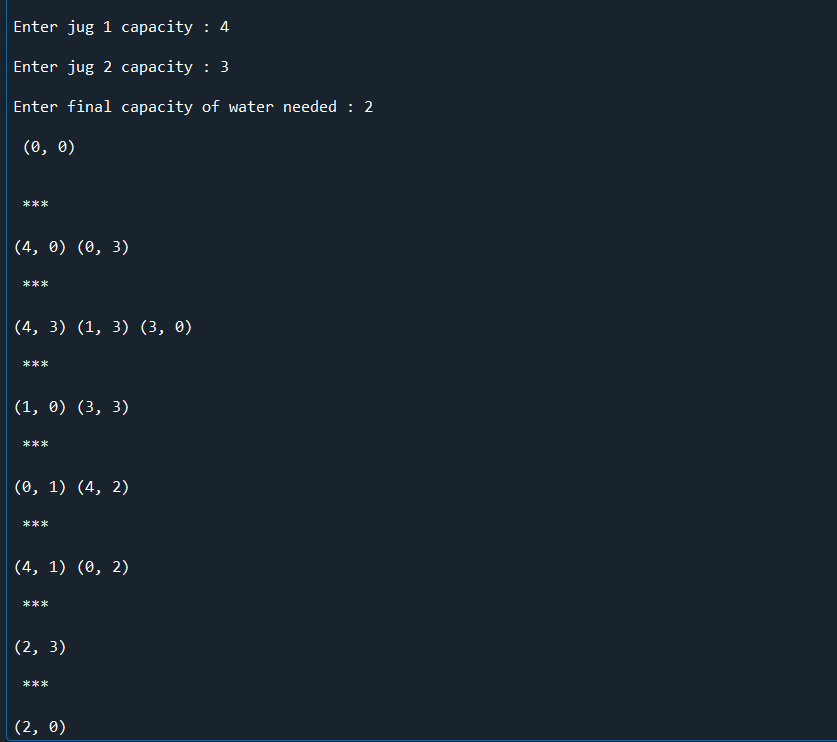
jug1 = int(input("Enter jug 1 capacity : "))

jug2 = int(input("Enter jug 2 capacity : "))

water = int(input("Enter final capacity of water needed : "))

water\_jug\_bfs(jug1, jug2, water)

**Output:**

****

**Result:**

Thus the rule based system (i.e) Water Jug Problem is implemented using bfs and State\_space (bfs).