EXPERIMENT 3:

AIM: To write the python program for Water Jug Problem

ALGORITHM:

1. **Solution Function**:
   * Define a function **Solution** that takes three parameters: **a** (capacity of jug 1), **b** (capacity of jug 2), and **target** (desired amount of water).
   * Initialize an empty dictionary **m** to store visited states.
   * Initialize a boolean variable **isSolvable** to track if a solution is found.
   * Initialize an empty list **path** to store the path from the initial state to the solution state.
   * Initialize a deque **q** to perform breadth-first search (BFS).
   * Append the initial state **(0, 0)** to the queue.
   * Perform BFS:
     + Pop the leftmost element **u** from the queue.
     + If **u** is already visited, continue to the next iteration.
     + If **u** is out of bounds, continue to the next iteration.
     + Append **u** to the **path** list and mark it as visited in dictionary **m**.
     + If **target** amount of water is reached in either jug, set **isSolvable** to True and break the loop.
     + Enqueue possible next states by pouring water from one jug to another and pouring out water from either jug.
   * If a solution is found, print the path from the initial state to the solution state.
   * If no solution is found, print "Solution not possible".
2. **Main Function**:
   * In the main block:
     + Define capacities of jugs **Jug1**, **Jug2**, and the **target** amount of water.
     + Print the message indicating the start of the solution path.
     + Call the **Solution** function with the provided capacities and target amount.

CODE:

from collections import deque

def Solution(a, b, target):

m = {}

isSolvable = False

path = []

q = deque()

q.append((0, 0))

while (len(q) > 0):

u = q.popleft()

if ((u[0], u[1]) in m):

continue

if ((u[0] > a or u[1] > b or

u[0] < 0 or u[1] < 0)):

continue

path.append([u[0], u[1]])

m[(u[0], u[1])] = 1

if (u[0] == target or u[1] == target):

isSolvable = True

if (u[0] == target):

if (u[1] != 0):

path.append([u[0], 0])

else:

if (u[0] != 0):

path.append([0, u[1]])

sz = len(path)

for i in range(sz):

print("(", path[i][0], ",",

path[i][1], ")")

break

q.append([u[0], b])

q.append([a, u[1]])

for ap in range(max(a, b) + 1):

c = u[0] + ap

d = u[1] - ap

if (c == a or (d == 0 and d >= 0)):

q.append([c, d])

c = u[0] - ap

d = u[1] + ap

if ((c == 0 and c >= 0) or d == b):

q.append([c, d])

q.append([a, 0])

q.append([0, b])

if (not isSolvable):

print("Solution not possible")

if \_\_name\_\_== '\_\_main\_\_':

Jug1, Jug2, target = 4, 3, 2

print("Path from initial state "

"to solution state ::")

Solution(Jug1, Jug2, target)

INPUT: Jug1 = 4

Jug2 = 3

target = 2

OUTPUT:

Path from initial state to solution state ::

( 0 , 0 )

( 0 , 3 )

( 4 , 0 )

( 4 , 3 )

( 3 , 0 )

( 1 , 3 )

( 3 , 3 )

( 4 , 2 )

( 0 , 2 )