In [ ]:

```
from collections import deque, defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # A function used by DFS
    def DFSUtil(self, v, visited):
        # Mark the current node as visited and print it
        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses recursive DFSUtil()
    def DFS(self, v):
        # Create a set to store visited vertices
        visited = set()

        # Call the recursive helper function to print DFS traversal
        self.DFSUtil(v, visited)

# Function to solve the water jug problem
def water_jug_solution(a, b, target):
    m = {}
    isSolvable = False
    path = []
    q = deque()
    q.append((0, 0))

    while q:
        u = q.popleft()
        if u 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] = 1

        if u[0] == target or u[1] == target:
            isSolvable = True
            if u[0] == target and u[1] != 0:
                path.append([u[0], 0])
            elif u[1] == target and u[0] != 0:
                path.append([0, u[1]])
```

```python
            for step in path:
                print(f"({step[0]}, {step[1]})")
            return

        q.append((u[0], b))  # Fill Jug2
        q.append((a, u[1]))  # Fill Jug1

        for ap in range(max(a, b) + 1):
            c, d = u[0] + ap, u[1] - ap
            if c == a or (d == 0 and d >= 0):
                q.append((c, d))

            c, d = u[0] - ap, u[1] + ap
            if (c == 0 and c >= 0) or d == b:
                q.append((c, d))

        q.append((a, 0))  # Empty Jug1
        q.append((0, b))  # Empty Jug2

    print("Solution not possible")

# Driver code
if __name__ == "__main__":
    g = Graph()
    g.addEdge(0, 1)
    g.addEdge(0, 2)
    g.addEdge(1, 2)
    g.addEdge(2, 0)
    g.addEdge(2, 3)
    g.addEdge(3, 3)

    print("Following is Depth First Traversal (starting from vertex 2)")
    g.DFS(2)

    print("\nSolving Water Jug Problem:")
    Jug1, Jug2, target = 4, 3, 2
    water_jug_solution(Jug1, Jug2, target)
```