```python
import heapq

# Define the state class to represent the current state of the jugs
class State:
    def __init__(self, jug1, jug2, parent=None, action="Initial"):
        self.jug1 = jug1
        self.jug2 = jug2
        self.parent = parent
        self.action = action
        self.g = 0
        if parent:
            self.g = parent.g + 1

    def __lt__(self, other):
        h_self = min(abs(self.jug1 - target), abs(self.jug2 - target))
        h_other = min(abs(other.jug1 - target), abs(other.jug2 - target))
        return (self.g + h_self) < (other.g + h_other)

    def __eq__(self, other):
        return self.jug1 == other.jug1 and self.jug2 == other.jug2

    def __hash__(self):
        return hash((self.jug1, self.jug2))


# A* algorithm
def solve_water_jug_problem(capacity1, capacity2, target):

    initial_state = State(0, 0)
    open_list = []
    heapq.heappush(open_list, initial_state)

    closed_set = set()

    while open_list:

        current_state = heapq.heappop(open_list)

        if current_state.jug1 == target or current_state.jug2 == target:

            path = []

            while current_state.parent:
                path.append(current_state.action)
```

```
            current_state = current_state.parent

        path.reverse()
        return path

    if (current_state.jug1, current_state.jug2) in closed_set:
        continue

    closed_set.add((current_state.jug1, current_state.jug2))


    # Fill jug1
    if current_state.jug1 < capacity1:
        heapq.heappush(open_list,
            State(capacity1, current_state.jug2, current_state,
            "Fill jug1 Completely"))


    # Fill jug2
    if current_state.jug2 < capacity2:
        heapq.heappush(open_list,
            State(current_state.jug1, capacity2, current_state,
            "Fill jug2 Completely"))


    # Empty jug1
    if current_state.jug1 > 0:
        heapq.heappush(open_list,
            State(0, current_state.jug2, current_state,
            "Empty jug1"))


    # Empty jug2
    if current_state.jug2 > 0:
        heapq.heappush(open_list,
            State(current_state.jug1, 0, current_state,
            "Empty jug2"))


    # Pour jug1 to jug2
    if current_state.jug1 > 0 and current_state.jug2 < capacity2:

        pour = min(current_state.jug1, capacity2 - current_state.jug2)
```

```
            heapq.heappush(open_list,
                State(current_state.jug1 - pour,
                current_state.jug2 + pour,
                current_state,
                f"Pour jug1 to jug2 ({pour}L)"))


        # Pour jug2 to jug1
        if current_state.jug2 > 0 and current_state.jug1 < capacity1:

            pour = min(current_state.jug2, capacity1 - current_state.jug1)

            heapq.heappush(open_list,
                State(current_state.jug1 + pour,
                current_state.jug2 - pour,
                current_state,
                f"Pour jug2 to jug1 ({pour}L)"))

    return None


# Main
def main():

    global target

    capacity1 = 4
    capacity2 = 3
    target = 2

    solution = solve_water_jug_problem(capacity1, capacity2, target)

    if solution:

        print("Solution found in", len(solution), "steps")

        for i, step in enumerate(solution, 1):
            print("Step", i, ":", step)

    else:
        print("No solution found")


if __name__ == "__main__":
```

```
    main()
```

Output:

Solution found in 4 steps
Step 1 : Fill jug2 Completely
Step 2 : Pour jug2 to jug1 (3L)
Step 3 : Fill jug2 Completely
Step 4 : Pour jug2 to jug1 (1L)

=== Code Execution Successful ===