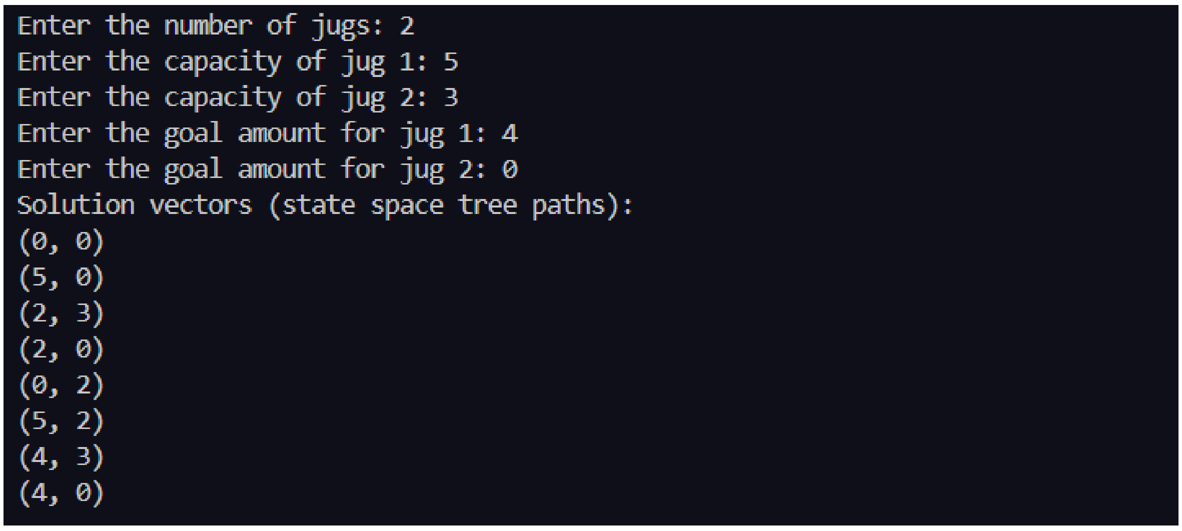
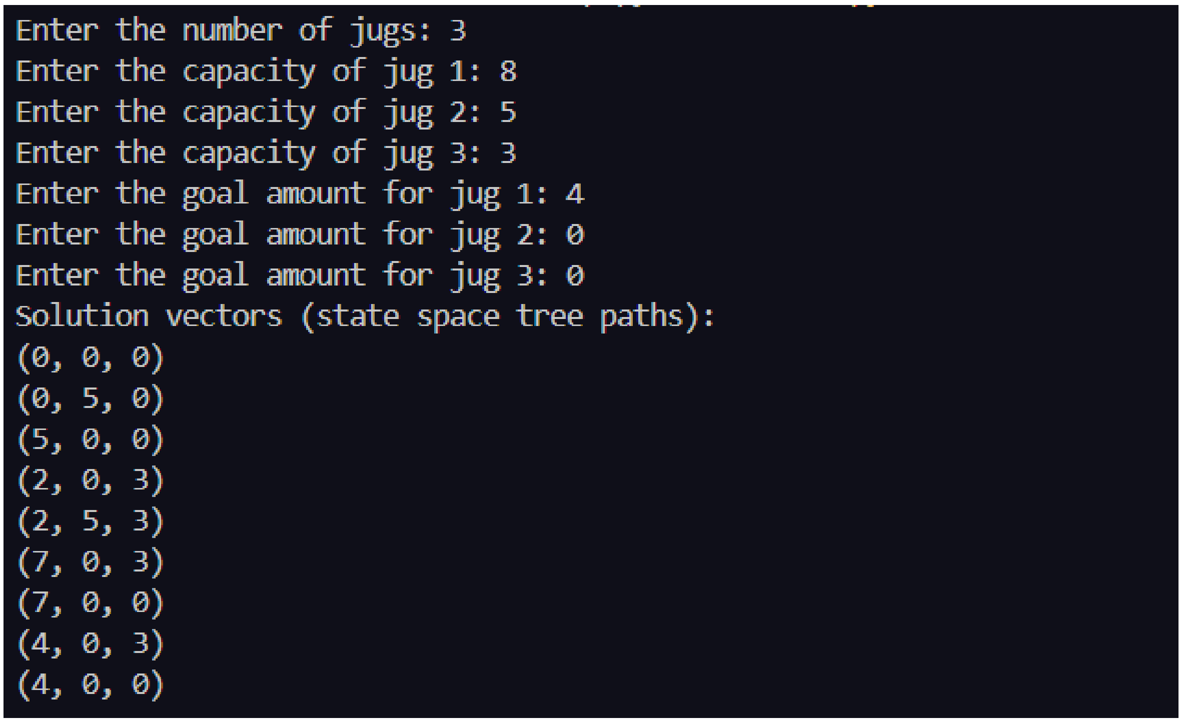
```
#############################################################
#                                                           #
#                       EXPERIMENT 3                        #
#                    Water Jug Problem                      #
#                 Nathan Cordeiro 22co09                    #
#                                                           #
#############################################################

from collections import deque

def get_inputs():
   num_jugs = int(input("Enter the number of jugs: "))
   jugs = []
   for i in range(num_jugs):
      capacity = int(input(f"Enter the capacity of jug {i+1}: "))
      jugs.append(capacity)
   goals = []
   for i in range(num_jugs):
      goal = int(input(f"Enter the goal amount for jug {i+1}: "))
      goals.append(goal)
   return jugs, goals

def is_goal(state, goals):
   return state == tuple(goals)

def get_next_states(state, jugs):
   next_states = []
   for i in range(len(jugs)):
      # Fill jug i
      new_state = list(state)
      new_state[i] = jugs[i]
      next_states.append(tuple(new_state))

      # Empty jug i
      new_state = list(state)
      new_state[i] = 0
      next_states.append(tuple(new_state))

      # Pour water from jug i to jug j
      for j in range(len(jugs)):
         if i != j:
            new_state = list(state)
            transfer = min(new_state[i], jugs[j] - new_state[j])
            new_state[i] -= transfer
            new_state[j] += transfer
            next_states.append(tuple(new_state))
```

```
    return next_states

def bfs(jugs, goals):
    initial_state = tuple([0] * len(jugs))
    queue = deque([initial_state])
    visited = set([initial_state])
    parent = {initial_state: None}
    solutions = []

    while queue:
        current_state = queue.popleft()
        if is_goal(current_state, goals):
            path = []
            temp_state = current_state
            while temp_state is not None:
                path.append(temp_state)
                temp_state = parent[temp_state]
            solutions.append(path[::-1])
            continue

        next_states = get_next_states(current_state, jugs)
        for next_state in next_states:
            if next_state not in visited:
                visited.add(next_state)
                parent[next_state] = current_state
                queue.append(next_state)

    return solutions

def main():
    jugs, goals = get_inputs()
    solutions = bfs(jugs, goals)
    if solutions:
        print("Solution vectors (state space tree paths):")
        for solution in solutions:
            for state in solution:
                print(state)
            print()
    else:
        print("No solution found")

if __name__ == "__main__":
    main()
```

OUTPUT:



```
Enter the number of jugs: 2
Enter the capacity of jug 1: 5
Enter the capacity of jug 2: 3
Enter the goal amount for jug 1: 4
Enter the goal amount for jug 2: 0
Solution vectors (state space tree paths):
(0, 0)
(5, 0)
(2, 3)
(2, 0)
(0, 2)
(5, 2)
(4, 3)
(4, 0)
```



```
Enter the number of jugs: 3
Enter the capacity of jug 1: 8
Enter the capacity of jug 2: 5
Enter the capacity of jug 3: 3
Enter the goal amount for jug 1: 4
Enter the goal amount for jug 2: 0
Enter the goal amount for jug 3: 0
Solution vectors (state space tree paths):
(0, 0, 0)
(0, 5, 0)
(5, 0, 0)
(2, 0, 3)
(2, 5, 3)
(7, 0, 3)
(7, 0, 0)
(4, 0, 3)
(4, 0, 0)
```