INDIVIDUAL ASSIGNEMNT

NGUYEN DAC TIN – M11323801

I. Cannibals and Missionaries

1. Formal presentation

a. State representation

A state is represented as a tuple (M_right, C_right, M_left, C_left, B) where:

b. Start state

For n missionaries and cannibals: (n, n, 0, 0, 1)

Example for n=3: (3, 3, 0, 0, 1)

c. Goal state

For n missionaries and cannibals: (0, 0, n, n, 0)

Example for n=3: (0, 0, 3, 3, 0)

d. Operators (successor function)

The operators are the possible moves of the boat. For a boat capacity c, the possible moves are:

- Move 1 to c missionaries
- Move 1 to c cannibals
- Move a combination of missionaries and cannibals where $total \leq c$

The successor function generates all valid states resulting from these moves, ensuring:

- The number of people on the boat doesn't exceed its capacity
- The number of cannibals never exceeds the number of missionaries

on either bank (if missionaries are present)

- No negative numbers of people

e. State space

All possible valid combinations of (M_right, C_right, M_left, C_left, B) where:

- 0 ≤ M_right, C_right, M_left, C_left ≤ n
- M_right + M_left = n
- C_right + C_left = n
- B is either 0 or 1
- If M_right > 0, then M_right ≥ C_right
- If M_left > 0, then M_left ≥ C_left

f. Solution

A sequence of states from the start state to the goal state, where each transition represents a valid boat move.

g. State space tree
The state space tree would show:
- Root node: Start state (n, n, 0, 0, 1)
- Intermediate nodes: Valid states after each move
- Leaf nodes: Either the goal state or dead-end states
- Edges: Represent boat moves
- Path from root to goal state node: The solution

2. Heuristic approach

```
def heuristic(self, board):
    M_right, C_right = board[0], board[1]
    return M_right + C_right
```

This heuristic counts the total number of people (missionaries and cannibals) on the right bank. This heuristic is effective for the following reasons:
- Admissibility: It never overestimates the cost to the goal, as each move can transport at most the boat's capacity.
- Informativeness: States with fewer people on the right bank are generally closer to the goal state.
- Efficiency: It's computationally simple, allowing quick evaluation of many states.

3. Explanation on the choice of representation

The state as a numpy array: [M_right, C_right, M_left, C_left, boat_position] is efficient and effective for several key reasons:
- Compactness: It uses just five integers to fully describe the state, minimizing memory usage.
- Completeness: It captures all necessary information, including the boat's position.
- Efficiency: Numpy arrays allow for fast operations and comparisons, crucial for exploring many states quickly.
- Easy manipulation: Applying moves and checking validity involve simple arithmetic operations on array elements.
- Symmetry: It inherently captures the problem's symmetry, reducing redundant information.

4. Solution printout and Diagram of search tree

a. Screenshot of solution printout

```
Start with n = 3 and c = 2

Step 1:
Right Bank - M: 3, C: 3 | Left Bank - M: 0, C: 0 | Boat: Right
Valid Moves from Current State: [(0, 1), (0, 2), (1, 1)]

Step 2:
Right Bank - M: 2, C: 2 | Left Bank - M: 1, C: 1 | Boat: Left
Valid Moves from Current State: [(-1, -1)]

Step 3:
Right Bank - M: 1, C: 1 | Left Bank - M: 2, C: 2 | Boat: Right
Valid Moves from Current State: [(1, 0), (1, 1)]

Step 4:
Right Bank - M: 0, C: 0 | Left Bank - M: 3, C: 3 | Boat: Left
Solution found!
```

b. Diagram of search tree

```
              [3M, 3C, 0M, 0C, R] (1)
                      |
                      v
          [1M, 3C, 2M, 0C, L] (2)
              /                 \
             v                   v
   [3M, 3C, 0M, 0C, R] (3)   [2M, 2C, 1M, 1C, R] (4)
         |                            |
         v                            v
   [1M, 3C, 2M, 0C, L] (5)   [0M, 2C, 3M, 1C, L] (6)
                                      |
                                      v
                               [2M, 2C, 1M, 1C, R] (7)
                                      |
                                      v
                               [0M, 2C, 3M, 1C, L] (8)
                                      |
                                      v
                               [1M, 1C, 2M, 2C, R] (9)
                                      |
                                      v
                               [0M, 0C, 3M, 3C, L] (10) – Solution
```

II. Vase puzzle

1. Formal problem representation

a. State space

- Each state is represented by a 4-tuple: (R, V1, V2, V3).
- Constraint: R + V1 + V2 + V3 = 24 (total amount of balsam is always 24 ounces)

b. Initial state: [24, 0, 0, 0] (24 ounces remaining, all vessels empty)

c. Goal state: [0, 8, 8, 8] (no balsam remaining, 8 ounces in each vessel)

d. Operators (successor function)

The operators are the possible pouring actions:

- Pour from R to V1, V2, or V3
- Pour from V1, V2, or V3 to R
- Pour between V1, V2, and V3

The successor function generates all valid states resulting from these actions, ensuring:

- No vessel exceeds its capacity
- The total amount of balsam remains 24 ounces
- No negative amounts

e. State space

All possible valid combinations of (R, V1, V2, V3) where:
- $0 \leq R \leq 24$
- $0 \leq V1 \leq 5$
- $0 \leq V2 \leq 11$
- $0 \leq V3 \leq 13$
- $R + V1 + V2 + V3 = 24$

f. Solution

A sequence of states from the start state to the goal state, where each transition represents a valid pouring action.

g. State space tree

The state space tree would show:
- Root node: Start state (24, 0, 0, 0)
- Intermediate nodes: Valid states after each pouring action
- Leaf nodes: Either the goal state or dead-end states
- Edges: Represent pouring actions
- Path from root to goal state node: The solution

2. Heuristic approach

```
def heuristic(self, board):
    return abs(8 - board[1]) + abs(8 - board[2]) + abs(8 - board[3])
```

This heuristic calculates the total difference between the current amount in each vessel and the goal amount of 8 ounces. This heuristic is effective for the following reasons:
- Admissibility: It never overestimates the cost to the goal, as each pour action can at most reduce the difference by the amount poured.
- Informativeness: States with vessels closer to containing 8 ounces each are generally closer to the goal state.
- Efficiency: It's computationally simple, allowing quick evaluation of many states.

3. Explanation on the choice of representation

The state as a numpy array: [R, V1, V2, V3] is efficient and effective for several key reasons:

- Compactness: It uses just four integers to fully describe the state, minimizing memory usage.
- Completeness: It captures all necessary information, including the remaining balsam and the amount in each vessel.
- Efficiency: Numpy arrays allow for fast operations and comparisons, crucial for exploring many states quickly.
- Easy manipulation: Applying moves and checking validity involve simple arithmetic operations on array elements.
- Implicit constraints: The total amount of 24 ounces is implicitly maintained, reducing redundant information.

4. Solution printout and Diagram of search tree

a. Screenshot of solution print out

```
Vase Problem Solution Process:
==============================

Step 1:
Exploring State: [24  0  0  0]
Remaining: 24 | Vessel 1 (5oz): 0 | Vessel 2 (11oz): 0 | Vessel 3 (13oz): 0
Heuristic Value: 24
Valid Moves:
  1. From Remaining to Vessel 1: 5 oz
  2. From Remaining to Vessel 2: 11 oz
  3. From Remaining to Vessel 3: 13 oz
New states added: 3

Step 2:
Exploring State: [13  0 11  0]
Remaining: 13 | Vessel 1 (5oz): 0 | Vessel 2 (11oz): 11 | Vessel 3 (13oz): 0
Heuristic Value: 19
Valid Moves:
  1. From Remaining to Vessel 1: 5 oz
  2. From Remaining to Vessel 3: 13 oz
  3. From Vessel 2 to Remaining: 11 oz
  4. From Vessel 2 to Vessel 1: 5 oz
  5. From Vessel 2 to Vessel 3: 11 oz
New states added: 5
...
New states added: 0

==============================
No solution found.
```

b. Diagram of search tree

```
1. (24, 0, 0, 0) h=24
   |
   |-- 2. (19, 5, 0, 0) h=19
   |    |
   |    |-- 5. (19, 0, 5, 0) h=19
   |    |
   |    |-- 6. (19, 0, 0, 5) h=19
   |
   |-- 3. (13, 0, 11, 0) h=13
   |    |
   |    |-- 7. (13, 5, 6, 0) h=13
   |    |
   |    |-- 8. (13, 0, 6, 5) h=13
   |
   |-- 4. (11, 0, 0, 13) h=11
        |
        |-- 9. (11, 5, 0, 8) h=6 *
             |
             |-- 10. (6, 5, 5, 8) h=1 *
                  |
                  |-- 11. (0, 8, 8, 8) h=0 * [GOAL]
```

Key:

- Each node is represented as: `node_number. (R, V1, V2, V3)`
- h: Heuristic value (sum of differences from 8 in each vessel)
- * Nodes on the solution path
- [GOAL]: Goal state reached

* Problem with Current Heuristic:
The search terminates without finding a solution.
The current heuristic might not be guiding the search effectively towards the goal state. It may need refinement to better estimate the distance to the goal.