Practical 2

**Aim: Implement solution of Water Jug problem or 8 puzzle problem using Best First Search or A\*.**

# Solving the Water Jug Problem using Best-First Search

Here we will see how to solve the Water Jug Problem using the Best-First Search algorithm. The problem involves two jugs with given capacities, and we need to find a sequence of operations to measure an exact amount of water.

# Heuristic Function

The heuristic function helps guide the search by estimating how close we are to the target amount of water.

We use min(abs(state[0] - goal), abs(state[1] - goal)) instead of summing absolute differences.

This ensures we prioritize the state that is closest to achieving the goal.

|  |
| --- |
| **def** heuristic(state, goal):  **return** min(abs(state[0] **-** goal), abs(state[1] **-** goal)) |

In [1]:

# Best-First Search Algorithm

This function implements the Best-First Search for solving the Water Jug problem.

**Key Steps:**

1. **Use a Priority Queue to always expand the most promising state first (i.e., the one closest to the target).**
2. **Maintain a visited set to avoid processing duplicate states.**
3. **Perform the following operations at each step:**

**Fill** either jug completely.

**Empty** either jug completely.

**Pour water** from one jug to another without exceeding capacity.

1. **If a state reaches the target amount, return the path taken.**

|  |
| --- |
| **from**    queue    **import**    PriorityQueue      **def**  best\_first\_search  (  jug1  ,  jug2  ,  goal  ):    visited  **=**    set  ()    pq    **=**    PriorityQueue  ()    pq  **.**  put  ((  heuristic  ((  0  ,    0  )  ,    goal  )  ,    (  0  ,    0  )  ,    []))      **while**    **not**    pq  **.**  empty  ():    \_  ,    (  x  ,    y  )  ,    path    **=**    pq  **.**  get  ()      **if**    (  x  ,    y  )    **in**    visited  : |
|  |

In [2]:

## Define Problem Variables

We define the capacities of the two jugs and the target amount of water we need to measure.

|  |
| --- |
| jug1\_capacity **=** 4 jug2\_capacity **=** 3 target **=** 2 |

In [3]:

# Execute the Algorithm

We now run the best\_first\_search function with the defined jug capacities and target. The solution path will show the sequence of jug states leading to the goal.

|  |
| --- |
| solution **=** best\_first\_search(jug1\_capacity, jug2\_capacity, target) print("Solution Path:", solution) |

In [4]:

Solution Path: [(0, 0), (0, 3), (3, 0), (3, 3), (4, 2)]