🧪 Experiment Number: 2

📘 Subject: Artificial Intelligence

📚 Semester: 5th Semester

🎯 Aim

To write a program to implement the **Three-Jug Water Jug Problem** using **Breadth First Search (BFS)** for solving a state-space search problem.

🧠 Theory

Three-Jug Water Jug Problem Overview

The **3-jug water jug problem** is an extension of the classic 2-jug problem. It involves three jugs with given capacities and an unlimited supply of water. The objective is to measure an exact quantity of water in one of the jugs using a series of valid operations.

State Representation

Each state is represented as a triplet `(x, y, z)` where:

- `x` is the amount of water in Jug A
- `y` is the amount in Jug B
- `z` is the amount in Jug C

Allowed Operations

You can perform any of the following:

1. Fill any of the jugs completely.
2. Empty any of the jugs.
3. Pour water from one jug to another (until the source is empty or the destination is full).

For 3 jugs (A, B, C), this leads to a wide state space that must be explored efficiently.

Breadth First Search (BFS)

BFS is ideal here as it finds the **shortest sequence of operations** (least number of steps) to reach the goal state. BFS explores all possibilities level by level and avoids revisiting states using a visited set.

🧮 Algorithm (BFS for 3 Jugs)

1. Start with the initial state (usually `(0, 0, full)` if only the last jug is filled initially).
2. Use a queue to explore all valid operations level-wise.
3. At each step, generate all possible next states:
   - Fill any jug.
   - Empty any jug.
   - Pour water from one jug to another.
4. For each generated state, if it is unvisited, enqueue it and mark it visited.
5. Stop when any jug contains the **target amount**.

💻 Code (C++ Implementation)

```cpp
#include <iostream>
#include <queue>
#include <set>
#include <tuple>
using namespace std;

typedef tuple<int, int, int> State;

void printState(int a, int b, int c) {
    cout << "Current state: (" << a << ", " << b << ", " << c << ")\n";
}

bool bfsThreeJug(int capA, int capB, int capC, int target) {
    set<State> visited;
    queue<State> q;

    // Assuming jug C is initially full, others are empty
    q.push({0, 0, capC});

    while (!q.empty()) {
        auto [a, b, c] = q.front();
        q.pop();

        printState(a, b, c);
```

```
        if (a == target || b == target || c == target) {
            cout << "🎯 Goal reached: (" << a << ", " << b << ", " << c <<
")\n";
            return true;
        }

        if (visited.count({a, b, c}))
            continue;

        visited.insert({a, b, c});

        vector<State> nextStates;

        int amounts[3] = {a, b, c};
        int capacities[3] = {capA, capB, capC};

        // Pour from one jug to another
        for (int i = 0; i < 3; ++i) {
            for (int j = 0; j < 3; ++j) {
                if (i != j && amounts[i] != 0 && amounts[j] !=
capacities[j]) {
                    int from = amounts[i];
                    int to = amounts[j];
                    int pour = min(from, capacities[j] - to);

                    int newAmounts[3] = {amounts[0], amounts[1],
amounts[2]};
                    newAmounts[i] -= pour;
                    newAmounts[j] += pour;

                    nextStates.push_back({newAmounts[0], newAmounts[1],
newAmounts[2]});
                }
            }
        }

        // Fill each jug
        for (int i = 0; i < 3; ++i) {
            int newAmounts[3] = {amounts[0], amounts[1], amounts[2]};
            newAmounts[i] = capacities[i];
            nextStates.push_back({newAmounts[0], newAmounts[1],
newAmounts[2]});
        }

        // Empty each jug
        for (int i = 0; i < 3; ++i) {
            int newAmounts[3] = {amounts[0], amounts[1], amounts[2]};
            newAmounts[i] = 0;
            nextStates.push_back({newAmounts[0], newAmounts[1],
newAmounts[2]});
```

```
            }

            for (auto& state : nextStates) {
                if (!visited.count(state)) {
                    q.push(state);
                }
            }
        }

        cout << "❌ No solution found.\n";
        return false;
    }

    int main() {
        int jugA = 8;
        int jugB = 5;
        int jugC = 3;
        int target = 4;

        bfsThreeJug(jugA, jugB, jugC, target);

        return 0;
    }
```

🔢 Inputs/Outputs Example

Input in code:

```
int jugA = 8;
int jugB = 5;
int jugC = 3;
int target = 4;
```

Output:

```
Current state: (0, 0, 3)
Current state: (0, 3, 0)
...
🎯 Goal reached: (4, 0, X)
```

📈 Analysis of Code and Algorithm

- **Search Algorithm:** Breadth First Search (Uninformed)

- **Goal State:** Any jug having exactly the target amount
- **Cycle Detection:** Handled using a visited set of triplets

Complexities:

- **Time Complexity:** O(a *b* c) — product of all capacities (state space size)
- **Space Complexity:** O(a *b* c) — for visited states and queue

---

🌐 Real-Life Applications

- Fluid measurement in chemical and industrial processes
- Logic-based puzzle solving in games
- AI search strategy teaching and simulation
- Constraint satisfaction problems in robotics

✅ Conclusion

This experiment successfully implemented the **Three-Jug Water Jug Problem** using **Breadth First Search (BFS)**. It modeled the problem as a state-space and applied BFS to explore all valid states efficiently. The experiment highlights how classical search algorithms can solve real-world problems involving constraints and transitions.
