# LAB-1

**Question:**

Write Production rules to solve Water jug problem and basic introduction.

**Description:**

You have two jugs with different capacities and a goal to measure a specific amount of water using these jugs. The operations allowed are typically filling a jug, emptying a jug, or pouring water from one jug into the other until one is empty or the other is full.

- We will use search algorithms like Breadth-First Search (BFS) or Depth-First Search (DFS) to explore the state space.

- Represent each state as a pair of the amount of water in each jug.

- Keep track of visited states to avoid redundant operations.

- Use the operations to transition from one state to another and check if the goal state (desired amount of water) is reached.

**Source Code:**

#include <iostream>

#include <queue>

#include <set>

#include <utility>

#include <vector>

using namespace std;

typedef pair<int, int> State;

void printSequence(vector<State>& sequence) {

for (const auto& state : sequence) {

cout << "(" << state.first << ", " << state.second << ") ";

}

cout << endl;

}

bool waterJugBFS(int jug1, int jug2, int target) {

int maxJug1 = jug1;

int maxJug2 = jug2;

queue<vector<State>> q;

set<State> visited;

vector<State> initialSequence = {{0, 0}};

q.push(initialSequence);

visited.insert({0, 0});

while (!q.empty()) {

vector<State> sequence = q.front();

q.pop();

State current = sequence.back();

int curJug1 = current.first;

int curJug2 = current.second;

if (curJug1 == target || curJug2 == target) {

printSequence(sequence);

return true;

}

vector<State> nextStates;

nextStates.push\_back({maxJug1, curJug2});

nextStates.push\_back({curJug1, maxJug2});

nextStates.push\_back({0, curJug2});

nextStates.push\_back({curJug1, 0});

int pourIntoJug2 = min(curJug1, maxJug2 - curJug2);

nextStates.push\_back({curJug1 - pourIntoJug2, curJug2 + pourIntoJug2});

int pourIntoJug1 = min(curJug2, maxJug1 - curJug1);

nextStates.push\_back({curJug1 + pourIntoJug1, curJug2 - pourIntoJug1});

for (const auto& nextState : nextStates) {

if (visited.find(nextState) == visited.end()) {

vector<State> newSequence = sequence;

newSequence.push\_back(nextState);

q.push(newSequence);

visited.insert(nextState);

}

}

}

cout << "No solution found." << endl;

return false;

}

int main() {

int jug1, jug2, target;

cout << "Enter capacity of jug1: ";

cin >> jug1;

cout << "Enter capacity of jug2: ";

cin >> jug2;

cout << "Enter target amount: ";

cin >> target;

if (target > max(jug1, jug2)) {

cout << "Target amount is greater than both jug capacities." << endl;

return 1;

}

waterJugBFS(jug1, jug2, target);

return 0;

}

**Output:**

**1)**

Enter capacity of jug1: 4

Enter capacity of jug2: 3

Enter target amount: 2

(0, 0) (0, 3) (3, 0) (3, 3) (4, 2)

2)

Enter capacity of jug1: 6

Enter capacity of jug2: 4

Enter target amount: 3

No solution found.