VIT Vidyalankar Institute of Technology Accredited A+ by NAAC	**DEPARTMENT OF COMPUTER ENGINEERING**

VIT Vidyalankar Institute of Technology Accredited A+ by NAAC

Experiment No. 3B

Semester	T.E. Semester VI
Subject	ARTIFICIAL INTELLIGENCE (CSL 604)
Subject Professor In-charge	Prof. Avinash Shrivas
Assisting Teachers	Prof. Avinash Shrivas

Student Name	Deep Salunkhe
Roll Number	21102A0014
Lab Number	310A

Title:

Water jug problem solution using mathematical approach

Theory:

1. **Water Jug Problem**: The Water Jug Problem is a classic puzzle where the objective is to measure a specific quantity of water using jugs of known capacities. It involves pouring water between the jugs to achieve the desired amount. The problem can be approached mathematically using Diophantine equations, which express the relationship between the capacities of the jugs and the desired quantity of water.

2. **Diophantine Equations**: Diophantine equations are polynomial equations where the solutions are constrained to be integers. In the context of the Water Jug Problem, a Diophantine equation can be formulated to represent the relationship between the jug capacities and the desired quantity of water. By solving this equation, we can determine if it's possible to measure the desired quantity of water using the given jug capacities.

3. **Minimum Number of Steps**: To solve the Water Jug Problem efficiently, we can calculate the minimum number of steps required to measure the desired quantity of water. This involves simulating the pouring of water between the jugs while minimizing the number of operations. This can be achieved by implementing algorithms that optimize the pouring process to minimize the total number of step

Program Code:

```
#include<iostream>
#include <algorithm> // for min function
using namespace std;

int gcd(int x, int y) {
    if (y == 0)
        return x;
    else
        return gcd(y, x % y);
}

bool ispossible(int x, int y, int d) {
    int valgcd = gcd(x, y);
    if (d % valgcd == 0)
        return true;
    return false;
}

void fillx(int &xState, int &xcapacity) {
    xState = xcapacity;
}

void emptyy(int &yState, int ycapacity) {
    yState = 0;
}

void transfer_x_to_y(int &xState, int &yState, int &xcapacity, int &ycapacity) {
    int y_can_take = (ycapacity - yState);
    if (y_can_take >= xState) {
        yState = yState + xState;
        xState = 0;
    }
    else {
        xState = xState - y_can_take;
        yState = ycapacity;
    }
}

int pour(int fromCap, int toCap, int d) {
    int from = fromCap;
    int to = 0;
    int step = 1;

    while (from != d && to != d) {
        int temp = min(from, toCap - to);
```

```
        to += temp;
        from -= temp;
        step++;

        if (from == d || to == d)
            break;

        if (from == 0) {
            from = fromCap;
            step++;
        }

        if (to == toCap) {
            to = 0;
            step++;
        }
    }
    return step;
}

int minSteps(int m, int n, int d) {
    if (m > n)
        swap(m, n);

    if (d > n)
        return -1;

    if ((d % gcd(n, m)) != 0)
        return -1;

    return min(pour(n, m, d), pour(m, n, d));
}

int main() {
    int xcapacity, ycapacity, finalx;
    int xState = 0, yState = 0;
    bool possible = true;
    cout << "Enter capacity for first container" << endl;
    cin >> xcapacity;

    cout << "Enter capacity for second container" << endl;
    cin >> ycapacity;

    cout << "Enter Final state of first container" << endl;
    cin >> finalx;

    possible = ispossible(xcapacity, ycapacity, finalx);
```

```
    if (!possible) {
        cout << "Not possible" << endl;
        return 0;
    }

    int min_steps = minSteps(xcapacity, ycapacity, finalx);
    if (min_steps == -1) {
        cout << "Not possible" << endl;
        return 0;
    }

    cout << "Minimum number of steps required is " << min_steps << endl;
    //wasn't  able to find the approach in which we get solution in minmum step
    // calculated by Diophantine equation

    cout<<"X"<<" "<<"Y"<<endl;

    while(xState!=finalx){
        if(xState==0){
            fillx(xState,xcapacity);
        }else if(yState==ycapacity){
            emptyy(yState,ycapacity);
        }else{
            transfer_x_to_y(xState,yState,xcapacity,ycapacity);
        }

        cout<<xState<<" "<<yState<<endl;
        if(xState==finalx ||  yState==finalx ){
            cout<<"congratulation"<<endl;
            return 0;
        }

    }

    return 0;
}
```

Output:

```
PS E:\GIt\SEM-6\AI> cd "e:\GIt\SEM-6\AI\" ; if ($?) { g++ Lab3_B.cpp -o Lab3_B } ; if ($?) { .\Lab3_B }
Enter capacity for first container
4
Enter capacity for second container
3
Enter Final state of first container
2
Minimum number of steps required is 4
X Y
4 0
1 3
1 0
0 1
4 1
2 3
congratulation
PS E:\GIt\SEM-6\AI>
```

Conclusion:

The combined code integrates two approaches to solve the Water Jug Problem: the Diophantine equation approach and the minimum number of steps approach. By formulating and solving a Diophantine equation, the code determines if it's possible to measure the desired quantity of water using the given jug capacities. Additionally, it calculates the minimum number of steps required to achieve this goal efficiently.