//A\* water jug

import java.util.\*;

class State {

    int jugA, jugB;

    State parent;

    int cost;

    int heuristic;

    public State(int jugA, int jugB, State parent, int cost, int heuristic) {

        this.jugA = jugA;

        this.jugB = jugB;

        this.parent = parent;

        this.cost = cost;

        this.heuristic = heuristic;

    }

    // Check if two states are equal

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

        State state = (State) obj;

        return jugA == state.jugA && jugB == state.jugB;

    }

    // Hash code method to use State as key in HashSet

    @Override

    public int hashCode() {

        return Objects.hash(jugA, jugB);

    }

}

class PriorityQueueNode {

    State state;

    int priority;

    public PriorityQueueNode(State state, int priority) {

        this.state = state;

        this.priority = priority;

    }

}

class PriorityQueue {

    private List<PriorityQueueNode> nodes = new ArrayList<>();

    public void enqueue(State state, int priority) {

        nodes.add(new PriorityQueueNode(state, priority));

        nodes.sort(Comparator.comparingInt(n -> n.priority));

    }

    public State dequeue() {

        if (nodes.isEmpty()) return null;

        return nodes.remove(0).state;

    }

    public boolean isEmpty() {

        return nodes.isEmpty();

    }

    public void clear() {

        nodes.clear();

    }

}

public class Astarwaterjug {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the capacity of jug A: ");

        int jugACapacity = scanner.nextInt();

        System.out.print("Enter the capacity of jug B: ");

        int jugBCapacity = scanner.nextInt();

        System.out.print("Enter the target amount for jug B: ");

        int target = scanner.nextInt();

        scanner.close();

        solveWaterJug(jugACapacity, jugBCapacity, target);

    }

    public static void solveWaterJug(int jugACapacity, int jugBCapacity, int target) {

        State initialState = new State(0, 0, null, 0, heuristic(0, 0, target));

        PriorityQueue queue = new PriorityQueue();

        queue.enqueue(initialState, initialState.cost + initialState.heuristic);

        while (!queue.isEmpty()) {

            State currentState = queue.dequeue();

            if (isGoalState(currentState, target)) {

                printSolution(currentState);

                return;

            }

            generateNextStates(queue, currentState, jugACapacity, jugBCapacity, target);

        }

        System.out.println("Solution not found.");

    }

    public static boolean isGoalState(State state, int target) {

        return state.jugA == target || state.jugB == target;

    }

    public static int heuristic(int jugA, int jugB, int target) {

        return Math.abs(jugA - target) + Math.abs(jugB - target);

    }

    public static void generateNextStates(PriorityQueue queue, State current, int jugACapacity, int jugBCapacity, int target) {

        // Empty Jug A

        enqueueState(queue, new State(0, current.jugB, current, current.cost + 1, heuristic(0, current.jugB, target)));

        // Empty Jug B

        enqueueState(queue, new State(current.jugA, 0, current, current.cost + 1, heuristic(current.jugA, 0, target)));

        // Fill Jug A

        enqueueState(queue, new State(jugACapacity, current.jugB, current, current.cost + 1, heuristic(jugACapacity, current.jugB, target)));

        // Fill Jug B

        enqueueState(queue, new State(current.jugA, jugBCapacity, current, current.cost + 1, heuristic(current.jugA, jugBCapacity, target)));

        // Pour Jug A to Jug B

        int pourAmount = Math.min(current.jugA, jugBCapacity - current.jugB);

        enqueueState(queue, new State(current.jugA - pourAmount, current.jugB + pourAmount, current, current.cost + 1, heuristic(current.jugA - pourAmount, current.jugB + pourAmount, target)));

        // Pour Jug B to Jug A

        pourAmount = Math.min(current.jugB, jugACapacity - current.jugA);

        enqueueState(queue, new State(current.jugA + pourAmount, current.jugB - pourAmount, current, current.cost + 1, heuristic(current.jugA + pourAmount, current.jugB - pourAmount, target)));

    }

    public static void enqueueState(PriorityQueue queue, State state) {

        queue.enqueue(state, state.cost + state.heuristic);

    }

    public static void printSolution(State state) {

        if (state == null) return;

        printSolution(state.parent);

        System.out.println("Jug A: " + state.jugA + ", Jug B: " + state.jugB);

    }

}