//BFS missonaries and cannibals

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Queue;

import java.util.Set;

public class missandcannibals {

    public static int[][] moves = { {1, 1}, {2, 0}, {0, 2}, {0, 1}, {1, 0}};

    public static class State{

        int LM;

        int LC;

        boolean LB;

        int RM;

        int RC;

        boolean RB;

        State parent;

        public State(int LM, int LC, boolean LB, int RM, int RC, boolean RB, State parent){

            this.LM = LM;

            this.LC = LC;

            this.LB = LB;

            this.RM = RM;

            this.RC = RC;

            this.RB = RB;

            this.parent = parent;

        }

    }

    public static boolean isGoal(State cur, State goal){

        if(cur.LM != goal.LM || cur.LC != goal.LC || cur.LB != goal.LB || cur.RB != goal.RB || cur.RC != goal.RC || cur.RM != goal.RM) return false;

        return true;

    }

    public static List<String> way(State cur){

        List<String> path = new ArrayList<>();

        while (cur.parent != null) {

            State parent = cur.parent;

            System.out.println(parent.LM + " " + parent.LC + " " + parent.LB + " " + parent.RM + " " + parent.RC + " " + parent.RB);

            if(((parent.LM - cur.LM == 1) && (parent.LC - cur.LC == 1)) || ((parent.RM - cur.RM == 1) && (parent.RC - cur.RC == 1)))path.add("1M1C");

            else if(parent.LM - cur.LM == 2 || parent.RM - cur.RM == 2) path.add("2M");

            else if(parent.LC - cur.LC == 2 || parent.RC - cur.RC == 2) path.add("2C");

            else if(parent.LC - cur.LC == 1 || parent.RC - cur.RC == 1) path.add("1C");

            else if(parent.LM - cur.LM == 1 || parent.RM - cur.RM == 1) path.add("1M");

            cur = parent;

        }

        Collections.reverse(path);

        return path;

    }

    public static List<String> solution(State initial, State goal){

        Set<State> visited = new HashSet<>();

        Queue<State> queue = new LinkedList<>();

        queue.add(initial);

        visited.add(initial);

        while (!queue.isEmpty()) {

            State cur = queue.poll();

            if(isGoal(cur, goal)) return way(cur);

            if(cur.LB){

                for (int[] move : moves) {

                    int newLM = cur.LM - move[0];

                    int newLC = cur.LC - move[1];

                    int newRM = cur.RM + move[0];

                    int newRC = cur.RC + move[1];

                    if(newLC >= 0 && newLM >= 0 && newRC >= 0 && newRM >= 0){

                        if((newLM == 0 || newLM >= newLC) && (newRM == 0 || newRM >= newRC)){

                            State next = new State(newLM, newLC, false, newRM, newRC, true, cur);

                            if(!visited.contains(next)){

                                visited.add(next);

                                queue.add(next);

                            }

                        }

                    }

                }

            } else {

                for (int[] move : moves) {

                    int newLM = cur.LM + move[0];

                    int newLC = cur.LC + move[1];

                    int newRM = cur.RM - move[0];

                    int newRC = cur.RC - move[1];

                    if(newLC >= 0 && newLM >= 0 && newRC >= 0 && newRM >= 0){

                        if((newLM == 0 || newLM >= newLC) && (newRM == 0 || newRM >= newRC)){

                            State next = new State(newLM, newLC, true, newRM, newRC, false, cur);

                            if(!visited.contains(next)){

                                visited.add(next);

                                queue.add(next);

                            }

                        }

                    }

                }

            }

        }

        return null;

    }

    public static void main(String[] args) {

        State initial = new State(3, 3, true, 0, 0, false, null);

        State goal = new State(0, 0, false, 3, 3, true, null);

        System.out.println(solution(initial, goal));

    }

}