//DFS eight puzzle

import java.util.\*;

public class dfseightpuzzle {

    static int[][] moves = {{1, 0}, {0, -1}, {0, 1}, {-1, 0}}; // down, left, right, up

    static boolean isEqual(int[][] curr, int[][] goal) {

        for (int i = 0; i < goal.length; i++) {

            for (int j = 0; j < goal.length; j++) {

                if (curr[i][j] != goal[i][j]) {

                    return false;

                }

            }

        }

        return true;

    }

    static int[] findEmptyPos(int[][] curr) {

        for (int i = 0; i < curr.length; i++) {

            for (int j = 0; j < curr.length; j++) {

                if (curr[i][j] == 0) {

                    return new int[]{i, j};

                }

            }

        }

        return null;

    }

    static int[][] copy(int[][] curr) {

        int[][] next = new int[3][3];

        for (int i = 0; i < next.length; i++) {

            for (int j = 0; j < next.length; j++) {

                next[i][j] = curr[i][j];

            }

        }

        return next;

    }

    static String boardToString(int[][] board) {

        StringBuilder sb = new StringBuilder();

        for (int[] row : board) {

            for (int tile : row) {

                sb.append(tile);

            }

        }

        return sb.toString();

    }

    static int[][] stringToBoard(String str) {

        int[][] board = new int[3][3];

        int index = 0;

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                board[i][j] = str.charAt(index++) - '0';

            }

        }

        return board;

    }

    static List<String> way(Map<String, String> parentMap, String curr) {

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

        List<int[][]> matrixList = new ArrayList<>();

        matrixList.add(stringToBoard(curr));

        while (parentMap.get(curr) != null) {

            String parent = parentMap.get(curr);

            matrixList.add(stringToBoard(parent));

            int[] currEmptyPos = findEmptyPos(stringToBoard(curr));

            int[] parentEmptyPos = findEmptyPos(stringToBoard(parent));

            if (parentEmptyPos[0] - currEmptyPos[0] == 1) {

                path.add("Up");

            }

            if (parentEmptyPos[0] - currEmptyPos[0] == -1) {

                path.add("Down");

            }

            if (parentEmptyPos[1] - currEmptyPos[1] == 1) {

                path.add("Left");

            }

            if (parentEmptyPos[1] - currEmptyPos[1] == -1) {

                path.add("Right");

            }

            curr = parent;

        }

        Collections.reverse(path);

        Collections.reverse(matrixList);

        for (int[][] matrix : matrixList) {

            for (int[] row : matrix) {

                for (int tile : row) {

                    System.out.print(tile + " ");

                }

                System.out.println();

            }

            System.out.println();

        }

        return path;

    }

    static List<String> solution(int[][] initial, int[][] goal) {

        Stack<String> stack = new Stack<>();

        Map<String, String> parentMap = new HashMap<>();

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

        String initialStr = boardToString(initial);

        String goalStr = boardToString(goal);

        visited.add(initialStr);

        stack.add(initialStr);

        parentMap.put(initialStr, null);

        while (!stack.isEmpty()) {

            String currStr = stack.pop();

            int[][] curr = stringToBoard(currStr);

            if (currStr.equals(goalStr)) {

                return way(parentMap, currStr);

            }

            for (int[] move : moves) {

                int[] emptyPos = findEmptyPos(curr);

                int currEmptyPosRow = emptyPos[0];

                int currEmptyPosCol = emptyPos[1];

                int newEmptyPosRow = currEmptyPosRow + move[0];

                int newEmptyPosCol = currEmptyPosCol + move[1];

                if (newEmptyPosRow >= 0 && newEmptyPosRow < 3 && newEmptyPosCol >= 0 && newEmptyPosCol < 3) {

                    int[][] next = copy(curr);

                    next[currEmptyPosRow][currEmptyPosCol] = next[newEmptyPosRow][newEmptyPosCol];

                    next[newEmptyPosRow][newEmptyPosCol] = 0;

                    String nextStr = boardToString(next);

                    if (!visited.contains(nextStr)) {

                        visited.add(nextStr);

                        stack.add(nextStr);

                        parentMap.put(nextStr, currStr);

                    }

                }

            }

        }

        return null;

    }

    public static void main(String[] args) {

        int[][] initial = {{1, 2, 3},

                           {4, 5, 6},

                           {7, 8, 0}};

        int[][] goal = {{0, 1, 2},

                        {4, 5, 3},

                        {7, 8, 6}};

        List<String> solutionPath = solution(initial, goal);

        if (solutionPath != null) {

            System.out.println(solutionPath);

        } else {

            System.out.println("No solution found.");

        }

    }

}