//DLS eight puzzle

import java.util.\*;

public class dlseightpuzzle {

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

    public static int[] findEmptyPos(int[][] board) {

        for (var i = 0; i < board.length; i++) {

            for (int j = 0; j < board[i].length; j++) {

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

                    return new int[]{i, j};

                }

            }

        }

        return null;

    }

    public static int[][] copyBoard(int[][] board) {

        int[][] newBoard = new int[board.length][board[0].length];

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

            for (int j = 0; j < newBoard[i].length; j++) {

                newBoard[i][j] = board[i][j];

            }

        }

        return newBoard;

    }

    public static boolean isEqual(int[][] board1, int[][] board2) {

        for (int i = 0; i < board1.length; i++) {

            for (int j = 0; j < board1[i].length; j++) {

                if (board1[i][j] != board2[i][j]) {

                    return false;

                }

            }

        }

        return true;

    }

    public static List<String> solve(int[][] initial, int[][] goal, int limit) {

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

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

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

        String initialStr = Arrays.deepToString(initial);

        visited.add(initialStr);

        State initialState = new State(initial, 0, null, null);

        stack.add(initialState);

        parentMap.put(initialStr, null);

        while (!stack.isEmpty()) {

            State currentState = stack.pop();

            if (isEqual(currentState.board, goal)) {

                return constructPath(parentMap, currentState);

            }

            int[] emptyPos = findEmptyPos(currentState.board);

            if (currentState.depth + 1 <= limit) {

                for (int[] move : moves) {

                    int newRow = emptyPos[0] + move[0];

                    int newCol = emptyPos[1] + move[1];

                    if (newRow >= 0 && newRow < 3 && newCol >= 0 && newCol < 3) {

                        int[][] newBoard = copyBoard(currentState.board);

                        newBoard[emptyPos[0]][emptyPos[1]] = newBoard[newRow][newCol];

                        newBoard[newRow][newCol] = 0;

                        String newBoardStr = Arrays.deepToString(newBoard);

                        if (!visited.contains(newBoardStr)) {

                            visited.add(newBoardStr);

                            State newState = new State(newBoard, currentState.depth + 1, currentState, moveToString(move));

                            stack.add(newState);

                            parentMap.put(newBoardStr, newState);

                        }

                    }

                }

            }

        }

        return null;

    }

    public static String moveToString(int[] move) {

        if (move[0] == -1 && move[1] == 0) return "U";

        if (move[0] == 1 && move[1] == 0) return "D";

        if (move[0] == 0 && move[1] == -1) return "L";

        if (move[0] == 0 && move[1] == 1) return "R";

        return "";

    }

    public static List<String> constructPath(Map<String, State> parentMap, State goalState) {

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

        State currentState = goalState;

        while (currentState.parent != null) {

            path.add(currentState.move);

            currentState = currentState.parent;

        }

        Collections.reverse(path);

        return path;

    }

    public static void printBoard(int[][] board) {

        for (int[] row : board) {

            for (int tile : row) {

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

            }

            System.out.println();

        }

        System.out.println();

    }

    public static class State {

        int[][] board;

        int depth;

        State parent;

        String move;

        public State(int[][] board, int depth, State parent, String move) {

            this.board = board;

            this.depth = depth;

            this.parent = parent;

            this.move = move;

        }

    }

    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> solution = solve(initial, goal, 9);

        if (solution != null) {

            System.out.println("Solution found:");

            int[][] board = initial;

            printBoard(board);

            for (String move : solution) {

                System.out.println("Move: " + move);

                board = applyMove(board, move);

                printBoard(board);

            }

        } else {

            System.out.println("No solution found within the given depth limit.");

        }

    }

    public static int[][] applyMove(int[][] board, String move) {

        int[] emptyPos = findEmptyPos(board);

        int newRow = emptyPos[0], newCol = emptyPos[1];

        if (move.equals("U")) newRow++;

        else if (move.equals("D")) newRow--;

        else if (move.equals("L")) newCol++;

        else if (move.equals("R")) newCol--;

        if (newRow >= 0 && newRow < 3 && newCol >= 0 && newCol < 3) {

            int[][] newBoard = copyBoard(board);

            newBoard[emptyPos[0]][emptyPos[1]] = newBoard[newRow][newCol];

            newBoard[newRow][newCol] = 0;

            return newBoard;

        }

        return board;

    }

}