//A\* eight puzzle

import java.util.\*;

public class Astareightpuzzle

{

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

    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;

    }

    public static int calculateHeuristic(int[][] curr, int[][] goal)

        {

            int count=0;

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

            {

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

               {

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

                {

                    count++;

                }

               }

            }

            return count;

        }

    static List<String> way(Map<int[][], int[][]> parentMap, int[][] curr)

    {

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

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

        grid.add(curr);

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

        {

            int[][] parent=parentMap.get(curr);

            grid.add(parent);

            int[] CurrfindPos=FindEmptyPos(curr);

            int[] parentEmptyPos=FindEmptyPos(parent);

            if(parentEmptyPos[0]-CurrfindPos[0]==1)

            {

                path.add("Up");

            }

            if(parentEmptyPos[0]-CurrfindPos[0]==-1)

            {

                path.add("Down");

            }

            if(parentEmptyPos[1]-CurrfindPos[1]==1)

            {

                path.add("Left");

            }

            if(parentEmptyPos[1]-CurrfindPos[1]==-1)

            {

                path.add("Right");

            }

            curr=parent;

        }

            Collections.reverse(path);

            Collections.reverse(grid);

            for(int i=0; i<grid.size(); i++)

            {

                int[][] max=grid.get(i);

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

                {

                    for(int j1=0; j1<max.length; j1++)

                    {

                        System.out.print(max[j][j1]+" ");

                    }

                    System.out.println();

                }

                 System.out.println();

            }

            return path;

        }

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

    {

     Set<int[][]> visited=new HashSet<>();

      PriorityQueue<state> queue=new PriorityQueue<>();

      Map<int[][], int[][]> parentMap=new HashMap<>();

      visited.add(initial);

      queue.add(new state(initial, calculateHeuristic(initial, goal), 0));

      parentMap.put(initial, null);

      while(!queue.isEmpty())

      {

        state curr=queue.poll();

        if(isEqual(curr.matrix,goal))

        {

            return way(parentMap, curr.matrix);

        }

        for(int[] move: moves)

        {

           int[] emptyPos=FindEmptyPos(curr.matrix);

           int RowemptyPos=emptyPos[0];

           int ColemptyPos=emptyPos[1];

           int newEmptyPosRow=RowemptyPos+move[0];

           int newEmptyPosCol=ColemptyPos+move[1];

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

           {

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

            next[newEmptyPosRow][newEmptyPosCol]=0;

            next[RowemptyPos][ColemptyPos]=curr.matrix[newEmptyPosRow][newEmptyPosCol];

           if(!visited.contains(next))

           {

            queue.add(new state(next, calculateHeuristic(next, goal), curr.depth + 1));

            visited.add(next);

            parentMap.put(next, curr.matrix);

           }

        }

        }

      }

      return null;

    }

    public static class state implements Comparable<state>

    {

        int[][] matrix;

        int heuristic;

        int depth;

        state(int[][] grid, int heuristic, int depth)

        {

            this.matrix = grid;

            this.heuristic = heuristic;

            this.depth=depth;

        }

        public int compareTo(state other)

        {

            return Integer.compare(this.heuristic+this.depth, other.heuristic+this.depth);

        }

    }

    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}};

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

    }

}