

```

import java.util.*;

class State implements Comparable<State> {
    int[][] board;
    int x, y, level, cost;
    String path;

    State(int[][] b, int x, int y, int level, String path) {
        this.board = b;
        this.x = x;
        this.y = y;
        this.level = level;
        this.path = path;
        this.cost = level + heuristic(b);
    }

    static int heuristic(int[][] b) {
        int h = 0;
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                if (b[i][j] != 0)
                    h += Math.abs(i - (b[i][j] - 1) / 3) + Math.abs(j - (b[i][j] - 1) % 3);
        return h;
    }

    String flatten() {
        StringBuilder sb = new StringBuilder();
        for (int[] row : board)
            for (int val : row)
                sb.append(val).append(",");
        return sb.toString();
    }

    @Override
    public int compareTo(State o) {
        return Integer.compare(this.cost, o.cost);
    }
}

public class EightPuzzle {

    static int[][] copyBoard(int[][] b) {
        int[][] c = new int[3][3];
        for (int i = 0; i < 3; i++)

```

```

        c[i] = Arrays.copyOf(b[i], 3);
    return c;
}

static boolean isSolvable(int[][] b) {
    List<Integer> list = new ArrayList<>();
    for (int[] row : b)
        for (int val : row)
            list.add(val);
    int inv = 0;
    for (int i = 0; i < 9; i++)
        for (int j = i + 1; j < 9; j++)
            if (list.get(i) != 0 && list.get(j) != 0 && list.get(i) > list.get(j))
                inv++;
    return inv % 2 == 0;
}

static void solve(int[][] start) {
    int sx = 0, sy = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (start[i][j] == 0) {
                sx = i;
                sy = j;
            }

    PriorityQueue<State> pq = new PriorityQueue<>();
    Set<String> visited = new HashSet<>();
    pq.add(new State(start, sx, sy, 0, ""));

    int[] dx = {-1, 1, 0, 0}; // U, D, L, R
    int[] dy = {0, 0, -1, 1};
    char[] dir = {'U', 'D', 'L', 'R'};

    while (!pq.isEmpty()) {
        State curr = pq.poll();
        String key = curr.flatten();
        if (visited.contains(key)) continue;
        visited.add(key);

        if (isGoal(curr.board)) {
            System.out.println("Solved in " + curr.level + " moves");
            System.out.println("Moves: " + curr.path);
            printBoard(curr.board);
        }
    }
}

```

```

        return;
    }

    for (int i = 0; i < 4; i++) {
        int nx = curr.x + dx[i], ny = curr.y + dy[i];
        if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3) {
            int[][] newBoard = copyBoard(curr.board);
            newBoard[curr.x][curr.y] = newBoard[nx][ny];
            newBoard[nx][ny] = 0;
            pq.add(new State(newBoard, nx, ny, curr.level + 1, curr.path + dir[i]));
        }
    }
}

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

static boolean isGoal(int[][] b) {
    int val = 1;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++, val++)
            if (b[i][j] != (val % 9)) return false;
    return true;
}

static void printBoard(int[][] b) {
    for (int[] row : b) {
        for (int val : row)
            System.out.print((val == 0 ? "_" : val + " "));
        System.out.println();
    }
    System.out.println();
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int[][] start = new int[3][3];

    System.out.println("Enter 3x3 start state (use 0 for blank):");
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            start[i][j] = sc.nextInt();

    if (!isSolvable(start))
        System.out.println("Unsolvable puzzle.");
}

```

```
        else
            solve(start);
    }
}
//1 2 3
//4 0 6
//7 5 8
//input
```