PROBLEM STATEMENT :

Implement A star Algorithm for any game search problem.

PROGRAM :

Node.java

```
public class Node {

	public Node parent;
	public int[][] matrix;

	// Blank tile cordinates
	public int x, y;

	// Number of misplaced tiles
	public int cost;

	// The number of moves so far
	public int level;

	public Node(int[][] matrix, int x, int y, int newX, int newY, int level, Node parent) {
		this.parent = parent;
		this.matrix = new int[matrix.length][];
		for (int i = 0; i < matrix.length; i++) {
			this.matrix[i] = matrix[i].clone();
		}

		// Swap value
		this.matrix[x][y]	= this.matrix[x][y] + this.matrix[newX][newY];
		this.matrix[newX][newY] = this.matrix[x][y] - this.matrix[newX][newY];
		this.matrix[x][y]	= this.matrix[x][y] - this.matrix[newX][newY];

		this.cost = Integer.MAX_VALUE;
		this.level = level;
		this.x = newX;
		this.y = newY;
	}

}
```

Puzzle.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;

public class Puzzle {

    public int dimension = 3;

    // Bottom, left, top, right
    int[] row = { 1, 0, -1, 0 };
    int[] col = { 0, -1, 0, 1 };

    public int calculateCost(int[][] initial, int[][] goal) {
        int count = 0;
        int n = initial.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (initial[i][j] != 0 && initial[i][j] != goal[i][j]) {
                    count++;
                }
            }
        }
        return count;
    }

    public void printMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public boolean isSafe(int x, int y) {
        return (x >= 0 && x < dimension && y >= 0 && y < dimension);
    }

    public void printPath(Node root) {
        if (root == null) {
            return;
        }
        printPath(root.parent);
        printMatrix(root.matrix);
        System.out.println();
    }

    public boolean isSolvable(int[][] matrix) {
        int count = 0;
        List<Integer> array = new ArrayList<Integer>();
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                array.add(matrix[i][j]);
            }
        }

        Integer[] anotherArray = new Integer[array.size()];
        array.toArray(anotherArray);

        for (int i = 0; i < anotherArray.length - 1; i++) {
            for (int j = i + 1; j < anotherArray.length; j++) {
                if (anotherArray[i] != 0 && anotherArray[j] != 0 && anotherArray[i]
> anotherArray[j]) {
                    count++;
                }
            }
        }

        return count % 2 == 0;
    }

    public void solve(int[][] initial, int[][] goal, int x, int y) {
        PriorityQueue<Node> pq = new PriorityQueue<Node>(1000, (a, b) -> (a.cost +
a.level) - (b.cost + b.level));
        Node root = new Node(initial, x, y, x, y, 0, null);
        root.cost = calculateCost(initial, goal);
        pq.add(root);

        while (!pq.isEmpty()) {
            Node min = pq.poll();
            if (min.cost == 0) {
                printPath(min);
                return;
            }

            for (int i = 0; i < 4; i++) {
        if (isSafe(min.x + row[i], min.y + col[i])) {
            Node child = new Node(min.matrix, min.x, min.y, min.x + row[i], min.y +
col[i], min.level + 1, min);
            child.cost = calculateCost(child.matrix, goal);
            pq.add(child);
        }
       }
        }
    }

    public static void main(String[] args) {
        int[][] initial = { {1, 8, 2}, {0, 4, 3}, {7, 6, 5} };
        int[][] goal    = { {1, 2, 3}, {4, 5, 6}, {7, 8, 0} };

        // White tile coordinate
        int x = 1, y = 0;

        Puzzle puzzle = new Puzzle();
        if (puzzle.isSolvable(initial)) {
            puzzle.solve(initial, goal, x, y);
        }
        else {
            System.out.println("The given initial is impossible to solve");
        }
    }

}
```

OUTPUT :

```
1 8 2
0 4 3
7 6 5

1 8 2
4 0 3
7 6 5

1 0 2
4 8 3
7 6 5

1 2 0
4 8 3
7 6 5

1 2 3
4 8 0
7 6 5

1 2 3
4 8 5
7 6 0

1 2 3
4 8 5
7 0 6

1 2 3
4 0 5
7 8 6

1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0
```