```
import java.util.*;

class State {
   int[] state;
   int g;
   int f;

   public int getF() {
      return this.f;
   }

   State(int[] state, int g, int f) {
      this.state = state;
      this.g = g;
      this.f = f;
   }

   @Override
   public String toString() {
      return Arrays.toString(state);
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj) return true;
      if (!(obj instanceof State)) return false;
      State other = (State) obj;
      return Arrays.equals(this.state, other.state);
   }
```

```
    @Override
    public int hashCode() {
        return Arrays.hashCode(state);
    }
}

public class EightPuzzle {
    static int[] goalState = {1, 2, 3, 8, 0, 4, 7, 6, 5}; // Chuyển goalState thành mảng 1 chiều

    static int heuristic(int[] state) {
        int misplaced = 0;
        for (int i = 0; i < state.length; i++) {
            if (state[i] != goalState[i] && state[i] != 0) {
                misplaced++;
            }
        }
        return misplaced;
    }

    static List<int[]> generateSuccessors(int[] state) {
        List<int[]> successors = new ArrayList<>();
        int zeroIndex = -1;
        for (int i = 0; i < state.length; i++) {
            if (state[i] == 0) {
                zeroIndex = i;
                break;
            }
        }
```

```
    int[][] moves = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
    for (int[] move : moves) {
      int newIndex = zeroIndex + move[0] * 3 + move[1];
      if (newIndex >= 0 && newIndex < state.length) {
        int[] newState = state.clone(); // Copy array correctly
        newState[zeroIndex] = newState[newIndex];
        newState[newIndex] = 0;
        successors.add(newState);
      }
    }
    return successors;
  }

  static void printOpenList(List<State> openList, Map<State, Integer> stateNumberMap) {
    System.out.println("OPEN: [");
    if (openList.isEmpty()) {
      System.out.println("]\n");
      return;
    }
    for (State state : openList) {
      System.out.println("\t" + Arrays.toString(state.state) + "_f(S" + stateNumberMap.get(state) +
")=" +
          state.f + ",");
    }
    System.out.println("]\n");
  }

  static void printClosedSet(Set<State> closedSet, Map<State, Integer> stateNumberMap) {
    System.out.println("CLOSE: [");
```

```
    if (closedSet.isEmpty()) {
      System.out.println("]\n");
      return;
    }
    for (State state : closedSet) {
      System.out.println("\t" + Arrays.toString(state.state) + "_f(S" +
(state.equals(closedSet.iterator().next()) ? 0 : stateNumberMap.get(state)) + ")=" +
          state.f + ",");
    }
    System.out.println("]\n");
  }

  static int[][] aStar(int[] initial) {
    PriorityQueue<State> openList = new PriorityQueue<>(Comparator.comparingInt(s -> s.f));
    Map<State, Integer> stateNumberMap = new HashMap<>();
    Set<State> closedSet = new HashSet<>();

    State initialState = new State(initial, 0, heuristic(initial));
    openList.add(initialState); // Thêm trạng thái ban đầu vào openList
    stateNumberMap.put(initialState, 0); // Đánh số trạng thái ban đầu và thêm vào
stateNumberMap với số thứ tự là 0

    int stepNumber = 0;
    int stateNumber = 0;
    while (!openList.isEmpty()) {
      System.out.println("**Step: " + (stepNumber + 1));
      State currentState = openList.poll();
      int[] state = currentState.state;
      System.out.println("Current State: ");
```

```
        System.out.println(Arrays.toString(state));
        closedSet.add(currentState);
        System.out.println("Next State: \n");
        List<int[]> successors = generateSuccessors(state);
        List<State> tempStates = new ArrayList<>(); // Danh sách tạm thời

        for (int[] successor : successors) {
            int gSuccessor = currentState.g + 1;
            int fSuccessor = gSuccessor + heuristic(successor);
            State nextState = new State(successor, gSuccessor, fSuccessor);
            tempStates.add(nextState); // Thêm vào danh sách tạm thời
        }

        // Sắp xếp danh sách tạm thời dựa trên giá trị f của các trạng thái
        tempStates.sort(Comparator.comparingInt(State::getF));

        // Đánh số các trạng thái trong danh sách tạm thời
        for (State nextState : tempStates) {
            stateNumber++;
            stateNumberMap.put(nextState, stateNumber);
        }

        // Thêm các trạng thái đã được đánh số vào openList
        openList.addAll(tempStates);

        // In ra thông tin các trạng thái đã được sắp xếp và đánh số
        for (State nextState : tempStates) {
            System.out.println("\t" + Arrays.toString(nextState.state) + "_f(S" +
stateNumberMap.get(nextState) + ")=" + nextState.f + ",");
```

```
        }

        if (stateNumber == 0) {
            System.out.println("OPEN: [");
            System.out.println("\t" + Arrays.toString(currentState.state) + "_f(S" + (stateNumber + 1) +
")=" +
                    currentState.g + "+" + currentState.f + "=" + currentState.f + ",");
            System.out.println("]\n");
            System.out.println("CLOSE: []\n");
        } else {
            printOpenList(new ArrayList<>(openList), stateNumberMap);
            printClosedSet(closedSet, stateNumberMap);
        }

        if (Arrays.equals(state, goalState)) {
            return to2DArray(state);
        }
        closedSet.add(currentState);
        stepNumber++;
    }

    return null;
  }

  static int[][] to2DArray(int[] array) {
    int[][] result = new int[3][3];
    for (int i = 0; i < 3; i++) {
        System.arraycopy(array, i * 3, result[i], 0, 3);
    }
```

```
        return result;
    }

    public static void main(String[] args) {
        int[] initialState = {2, 8, 3, 1, 6, 4, 7, 0, 5};
        int[][] finalState = aStar(initialState);
        if (finalState != null) {
            System.out.println("Final State:");
            for (int[] row : finalState) {
                System.out.println(Arrays.toString(row));
            }
        } else {
            System.out.println("No solution found!");
        }
    }
}
```