```
import numpy as np
from queue import PriorityQueue

class State:
    def __init__(self, state, parent):
        self.state = state
        self.parent = parent

    def __lt__(self, other):
        return False

class Puzzle:
    def __init__(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state

    def is_goal(self, state):
        for i in range(3):
            for j in range(3):
                if state[i][j] != self.goal_state[i][j]:
                    return False
        return True

    def get_possible_moves(self, state):
        zero_pos = None
        for i in range(3):
            for j in range(3):
                if state[i][j] == 0:
                    zero_pos = (i, j)
        directions = [(0, 1), (0, -1), (-1, 0), (1, 0)]
        possible_moves = []
        for direction in directions:
            new_pos = (zero_pos[0] + direction[0], zero_pos[1] + direction[1])
            if 0 <= new_pos[0] < 3 and 0 <= new_pos[1] < 3:
                new_state = np.copy(state)
                new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
                possible_moves.append(new_state)
        return possible_moves

    def heuristic(self, state):
        return np.count_nonzero(state != self.goal_state)

    def print_state(self, state):
        print(state[:, :])

    def solve(self):
        queue = PriorityQueue()
        initial_state = State(self.initial_state, None)
        queue.put((0, initial_state))
        visited = set()
        while not queue.empty():
            priority, current_state = queue.get()
            if self.is_goal(current_state.state):
                return current_state
            for move in self.get_possible_moves(current_state.state):
                move_state = State(move, current_state)
                if str(move_state.state) not in visited:
                    visited.add(str(move_state.state))
                    priority = self.heuristic(move_state.state)
                    queue.put((priority, move_state))
        return None

initial_state = np.array([[2, 8, 1], [7, 4, 3], [6, 0, 5]])
goal_state = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])
puzzle = Puzzle(initial_state, goal_state)
solution = puzzle.solve()
moves = []
if solution is not None:
    while solution is not None:
        moves.append(solution.state)
        solution = solution.parent
    for move in reversed(moves):
        puzzle.print_state(move)
else:
    print("No solution found")
```

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

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