```python
class Puzzle:
    def __init__(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state
        self.rows = 3
        self.cols = 3

    def get_neighbors(self, state):

        zero_pos = [(i, j) for i in range(self.rows) for j in range(self.cols) if state[i][j] == 0][0]
        x, y = zero_pos

        # Possible directions to move the blank space: up, down, left, right
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        neighbors = []

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < self.rows and 0 <= new_y < self.cols:
                new_state = [list(row) for row in state]  # Create a copy of the state
                new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]  # Swap
                neighbors.append(new_state)

        return neighbors

    def dfs(self):
        stack = [(self.initial_state, [])]
        visited = set()

        while stack:
            current_state, path = stack.pop()

            # If we reached the goal, return the solution
            if current_state == self.goal_state:
                return path + [current_state]

            # Mark the current state as visited
            state_tuple = tuple(tuple(row) for row in current_state)
            if state_tuple not in visited:
                visited.add(state_tuple)

                # Explore all neighboring states
                for neighbor in self.get_neighbors(current_state):
                    stack.append((neighbor, path + [current_state]))

        return None

    def print_solution(self, solution):
        if solution:
            print("Solution found!")
            for step in solution:
                for row in step:
                    print(row)
                print()
        else:
            print("No solution exists.")

# Example usage
initial_state = [
    [1, 2, 3],
    [4, 0, 6],
    [7, 5, 8]
]

goal_state = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

```

```
puzzle = Puzzle(initial_state, goal_state)
solution = puzzle.dfs()
puzzle.print_solution(solution)
```

```
[0, 5, 4]

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

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

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

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

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

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

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

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

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

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

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

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

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

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