8-PUZZLE

```
from collections import deque

# Class representing the 8 Puzzle problem
class Puzzle8:
    def __init__(self, size=3):
        self.size = size

    def display_state(self, state):
        for i in range(self.size):
            for j in range(self.size):
                print(state[i * self.size + j], end=" ")
            print()

    def get_blank_index(self, state):
        return state.index(-1)

    def get_neighbors(self, state):
        neighbors = []
        blank_index = self.get_blank_index(state)
        row, col = divmod(blank_index, self.size)

        moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # right, down, left, up

        for move in moves:
            new_row, new_col = row + move[0], col + move[1]

            if 0 <= new_row < self.size and 0 <= new_col < self.size:
                new_state = state[:]
                new_blank_index = new_row * self.size + new_col

                # Swap the blank tile with the neighbor
                new_state[blank_index], new_state[new_blank_index] =
new_state[new_blank_index], new_state[blank_index]

                neighbors.append(new_state)

        return neighbors
```

```
    def is_goal_state(self, state, target_state):
        return state == target_state

    def bfs(self, initial_state, target_state):
        queue = deque([(initial_state, [])])
        visited = set()

        while queue:
            current_state, path = queue.popleft()

            if self.is_goal_state(current_state, target_state):
                return path

            if tuple(current_state) not in visited:
                visited.add(tuple(current_state))
                neighbors = self.get_neighbors(current_state)

                for neighbor in neighbors:
                    queue.append((neighbor, path + [neighbor]))

        return None

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

puzzle = Puzzle8()
solution = puzzle.bfs(initial_state, goal_state)

if solution:
    print("Solution found:")
    for step, state in enumerate(solution):
        print(f"Step {step + 1}:")
        puzzle.display_state(state)
        print()
else:
    print("No solution found.")
```

Output:

```
Solution found:
Step 1:
1 2 3
4 5 -1
6 7 8

Step 2:
1 2 3
4 5 8
6 7 -1

Step 3:
1 2 3
4 5 8
6 -1 7

Step 4:
1 2 3
4 5 8
-1 6 7

Step 5:
1 2 3
-1 5 8
4 6 7

Step 6:
1 2 3
5 -1 8
4 6 7

Step 7:
1 2 3
5 6 8
4 -1 7

Step 8:
1 2 3
5 6 8
4 7 -1

Step 9:
1 2 3
5 6 -1
4 7 8

Step 10:
1 2 3
5 -1 6
4 7 8
```

```
Step 10:
1 2 3
5 -1 6
4 7 8

Step 11:
1 2 3
-1 5 6
4 7 8

Step 12:
1 2 3
4 5 6
-1 7 8

Step 13:
1 2 3
4 5 6
7 -1 8

Step 14:
1 2 3
4 5 6
7 8 -1
```