```
from collections import deque

class PuzzleState:
    def __init__(self, board, empty_tile_pos, moves=0, previous=None):
        self.board = board
        self.empty_tile_pos = empty_tile_pos
        self.moves = moves
        self.previous = previous

    def is_goal(self):
        return self.board == [1, 2, 3, 4, 5, 6, 7, 8, 0]

    def get_possible_moves(self):
        possible_moves = []
        x, y = self.empty_tile_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Righ

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = self.board[:]
                # Swap the empty tile with the adjacent tile
                new_board[x * 3 + y], new_board[new_x * 3 + new_y] = new_board[
                possible_moves.append((new_board, (new_x, new_y)))

        return possible_moves

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

    while stack:
        state = stack.pop()
        if state.is_goal():
            return state
        visited.add(tuple(state.board))
        for new_board, new_empty_pos in state.get_possible_moves():
            new_state = PuzzleState(new_board, new_empty_pos, state.moves + 1, 
            if tuple(new_board) not in visited:
                stack.append(new_state)
    return None

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

    while queue:
        state = queue.popleft()
        if state.is_goal():
            return state
        visited.add(tuple(state.board))
        for new_board, new_empty_pos in state.get_possible_moves():
            new_state = PuzzleState(new_board, new_empty_pos, state.moves + 1,
            if tuple(new_board) not in visited:
                queue.append(new_state)
    return None

def print_solution(solution):
    path = []
    while solution:
        path.append(solution.board)
        solution = solution.previous
    for step in reversed(path):
        print(step[0:3])
        print(step[3:6])
        print(step[6:9])
        print()

def main():
    initial_board = [1, 2, 3, 4, 5, 6, 0, 7, 8]  # Example initial state
    empty_tile_pos = (2, 0)  # Position of the empty tile (0)

    initial_state = PuzzleState(initial_board, empty_tile_pos)

    # Solve with DFS
    print("Solving with DFS:")
    dfs_solution = dfs(initial_state)
    if dfs_solution:
        print("Solution found in", dfs_solution.moves, "moves:")
        print_solution(dfs_solution)
    else:
        print("No solution found.")

    # Solve with BFS
    print("Solving with BFS:")
    bfs_solution = bfs(initial_state)
    if bfs_solution:
        print("Solution found in", bfs_solution.moves, "moves:")
        print_solution(bfs_solution)
    else:
        print("No solution found.")

if __name__ == "__main__":
    main()
```

```
Solving with DFS:
Solution found in 2 moves:
[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]

Solving with BFS:
Solution found in 2 moves:
[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]
```