```
from collections import deque
GOAL_STATE = [[1, 2, 3], [4, 5, 6], [7, 8, None]]
MOVES = {
    'up': (-1, 0),
    'down': (1, 0),
    'left': (0, -1),
    'right': (0, 1),
}
```

```
def generate_successors(state):
    successors = []
    for i in range(3):
        for j in range(3):
            if state[i][j] is None:
                blank_pos = (i, j)
                break

    for direction, (di, dj) in MOVES.items():
        new_i, new_j = blank_pos[0] + di, blank_pos[1] + dj
        if 0 <= new_i < 3 and 0 <= new_j < 3:
            new_state = [row[:] for row in state]
            new_state[blank_pos[0]][blank_pos[1]], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[blank_pos[0]][blank_pos[1
            successors.append(new_state)

    return successors

def bfs(initial_state):
    queue = deque([(initial_state, [initial_state])])
    visited = set()
    visited.add(tuple(map(tuple, initial_state)))

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

        if current_state == GOAL_STATE:
            return path

        for next_state in generate_successors(current_state):
            state_tuple = tuple(map(tuple, next_state))
            if state_tuple not in visited:
                visited.add(state_tuple)
                queue.append((next_state, path + [next_state]))

    return None

def print_state(state):
    for row in state:
        print(' '.join(str(x) if x is not None else '_' for x in row))
    print()

if __name__ == "__main__":
    initial_state = [[1, 2, 3], [None, 4, 6], [7, 5, 8]]

    bfs_path = bfs(initial_state)
    print("BFS Solution Found:")
    if bfs_path:
        for step in bfs_path:
            print_state(step)
        print(f"Number of moves: {len(bfs_path) - 1}")
    else:
        print("No solution")
```

```
BFS Solution Found:
1 2 3
_ 4 6
7 5 8

1 2 3
4 _ 6
7 5 8

1 2 3
4 5 6
7 _ 8

1 2 3
4 5 6
7 8 _

Number of moves: 3
```

```
GOAL_STATE = [[1, 2, 3], [4, 5, 6], [7, 8, None]]
MOVES = [(-1, 0), (1, 0), (0, -1), (0, 1)]

def generate_successors(state):
    successors = []
    blank_i, blank_j = next((i, j) for i in range(3) for j in range(3) if state[i][j] is None)

    for di, dj in MOVES:
        new_i, new_j = blank_i + di, blank_j + dj
        if 0 <= new_i < 3 and 0 <= new_j < 3:
            new_state = [row[:] for row in state]
            new_state[blank_i][blank_j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[blank_i][blank_j]
            successors.append(new_state)

    return successors

def dfs(initial_state):
    stack = [(initial_state, 0)]
    visited = {tuple(map(tuple, initial_state))}

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

        if current_state == GOAL_STATE:
            print_state(current_state)
            return moves

        for next_state in generate_successors(current_state):
            state_tuple = tuple(map(tuple, next_state))
            if state_tuple not in visited:
                visited.add(state_tuple)
                stack.append((next_state, moves + 1))
    return -1

def print_state(state):
    print("Goal State Reached:")
    for row in state:
        print(' '.join(str(x) if x is not None else '_' for x in row))
    print()

if __name__ == "__main__":
    initial_state = [[1, 2, 3], [None, 4, 6], [7, 5, 8]]
    moves_to_goal = dfs(initial_state)
    if moves_to_goal != -1:
        print("DFS Solution Found: True")
        print(f"Number of moves to reach the goal state: {moves_to_goal}")
    else:
        print("DFS Solution Found: False")
```

```
Goal State Reached:
1 2 3
4 5 6
7 8 _

DFS Solution Found: True
Number of moves to reach the goal state: 49285
```