```python
from collections import deque

def print_board(state):
    for row in state:
        print(' '.join(str(x) for x in row))
    print()

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

def get_neighbors(state):
    neighbors = []
    x, y = [(i, row.index(0)) for i, row in enumerate(state) if 0 in row][0]
    moves = [(x + dx, y + dy) for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]]

    for nx, ny in moves:
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(new_state)

    return neighbors

def dfs(state, path, visited):
    if is_goal(state):
        return path
    visited.add(str(state))
    for neighbor in get_neighbors(state):
        if str(neighbor) not in visited:
            result = dfs(neighbor, path + [neighbor], visited)
            if result:
                return result
    visited.remove(str(state))
    return None

def solve_puzzle(start):
    return dfs(start, [], set())

if __name__ == "__main__":
    start_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]
    solution = solve_puzzle(start_state)
```

```
if solution:
    print("Solution path:")
    for step in solution:
        print_board(step)
else:
    print("No solution found.")
```

```
Solution path:
1 0 3
4 2 6
7 5 8

0 1 3
4 2 6
7 5 8

4 1 3
0 2 6
7 5 8

4 1 3
7 2 6
0 5 8

4 1 3
7 2 6
5 0 8

4 1 3
7 0 6
5 2 8

4 0 3
7 1 6
5 2 8

0 4 3
7 1 6
5 2 8

7 4 3
0 1 6
5 2 8

7 4 3
5 1 6
0 2 8

7 4 3
5 1 6
2 0 8

7 4 3
5 0 6
2 1 8

0 7 3
5 4 6
2 1 8

5 7 3
0 4 6
2 1 8

5 7 3
2 4 6
0 1 8

5 7 3
2 4 6
1 0 8

5 7 3
2 0 6
1 4 8

5 0 3
2 7 6
1 4 8

0 5 3
2 7 6
1 4 8

2 5 3
0 7 6
1 4 8

2 5 3
1 7 6
0 4 8

2 5 3
1 7 6
4 0 8

2 5 3
1 0 6
4 7 8

2 0 3
1 5 6
4 7 8

0 2 3
1 5 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
```