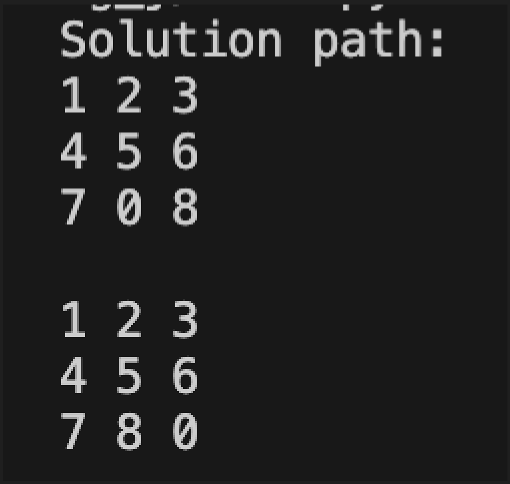
```python
from collections import deque

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 bfs(start):
   queue = deque([(start, [])])
   visited = set()

   while queue:
       state, path = queue.popleft()
       if is_goal(state):
           return path
       visited.add(str(state))
       for neighbor in get_neighbors(state):
           if str(neighbor) not in visited:
               queue.append((neighbor, path + [neighbor]))

   return None

def solve_puzzle(start):
   return bfs(start)

if __name__ == "__main__":
```

```python
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 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0
```