8 puzzle problem(non heuristic-BFS)

```
from collections import deque
class PuzzleState:
    def __init__(self, board, zero_pos, path):
        self.board = board
        self.zero_pos = zero_pos  # position of the empty tile (0)
        self.path = path  # path taken to reach this state

    def __str__(self):
        return "\n".join(
            [" ".join(map(str, self.board[i * 3 : (i + 1) * 3])) for i in range(3)]
        )


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


def get_neighbors(state):
    neighbors = []
    zero_row, zero_col = divmod(state.zero_pos, 3)
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # up, down, left, right

    for dr, dc in moves:
        new_row, new_col = zero_row + dr, zero_col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_zero_pos = new_row * 3 + new_col
            new_board = state.board[:]
            # Swap the zero with the adjacent tile
            new_board[state.zero_pos], new_board[new_zero_pos] = (
                new_board[new_zero_pos],
                new_board[state.zero_pos],
            )
            neighbors.append(
                PuzzleState(new_board, new_zero_pos, state.path + [new_board])
            )

    return neighbors


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

```
    visited.add(tuple(initial_state.board))

    while queue:
        current_state = queue.popleft()

        if is_goal(current_state.board):
            return current_state.path  # Return the path to the goal state

        for neighbor in get_neighbors(current_state):
            if tuple(neighbor.board) not in visited:
                visited.add(tuple(neighbor.board))
                queue.append(neighbor)

    return None  # Return None if no solution is found


if __name__ == "__main__":
    # Example initial state
    initial_board = [1, 2, 3, 0, 4, 5, 6, 7, 8]
    zero_pos = initial_board.index(0)
    initial_state = PuzzleState(initial_board, zero_pos, [])

    solution_path = bfs(initial_state)

    if solution_path:
        print("Stepwise moves to the goal state:\n")
        for step in solution_path:
            print(PuzzleState(step, step.index(0), []))
            print()  # Adding a blank line for space
        print("Goal state reached:\n")
        print(PuzzleState(solution_path[-1], 0, []))
    else:
        print("No solution found.")
```

OUTPUT:

```
Stepwise moves to the goal state:

1 2 3
4 0 5
6 7 8

1 2 3
4 5 0
6 7 8

1 2 3
4 5 8
6 7 0

1 2 3
4 5 8
6 0 7

1 2 3
4 5 8
0 6 7

1 2 3
0 5 8
4 6 7

1 2 3
5 0 8
4 6 7

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

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

1 2 3
5 6 0
4 7 8

1 2 3
5 0 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
```

```
Goal state reached:

1 2 3
4 5 6
7 8 0


...Program finished with exit code 0
Press ENTER to exit console.
```