$Solve\ 8 - puzzle\ problem\ using\ DFS\ and\ BFS\ algorithms$

```
from collections import deque

def is_solvable(state):
    inv_count = 0
    state_flat = [tile for row in state for tile in row if tile != 0]
    for i in range(len(state_flat)):
        for j in range(i + 1, len(state_flat)):
            if state_flat[i] > state_flat[j]:
                inv_count += 1
    return inv_count % 2 == 0

def find_blank(state):
    for i, row in enumerate(state):
        for j, val in enumerate(row):
            if val == 0:
                return i, j

def get_neighbors(state):
    neighbors = []
    blank_i, blank_j = find_blank(state)
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    for di, dj in directions:
        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]
            neighbors.append(new_state)
    return neighbors

def dfs(initial_state, goal_state):
    stack = [(initial_state, [])]
    visited = set()
    while stack:
        state, path = stack.pop()
        state_tuple = tuple(tuple(row) for row in state)
        if state_tuple in visited:
            continue
        visited.add(state_tuple)
        if state == goal_state:
            return path + [state]
        for neighbor in get_neighbors(state):
            stack.append((neighbor, path + [state]))
    return None

def bfs(initial_state, goal_state):
    queue = deque([(initial_state, [])])
    visited = set()
    while queue:
        state, path = queue.popleft()
        state_tuple = tuple(tuple(row) for row in state)
        if state_tuple in visited:
            continue
        visited.add(state_tuple)
        if state == goal_state:
            return path + [state]
        for neighbor in get_neighbors(state):
            queue.append((neighbor, path + [state]))
    return None

def display_path(path):
    for step, state in enumerate(path):
        print(f"Step {step}:")
        for row in state:
            print(row)
        print()

if __name__ == "__main__":
    print("Output: 1BM22CS290")
    print("Enter the initial state (3x3 grid, 0 for blank):")
    initial_state = [list(map(int, input().split())) for _ in range(3)]

    print("Enter the goal state (3x3 grid, 0 for blank):")
    goal_state = [list(map(int, input().split())) for _ in range(3)]
```

```
    if not is_solvable(initial_state):
        print("The given puzzle is not solvable.")
    else:
        print("Choose the method to solve the puzzle:")
        print("1. Depth-First Search (DFS)")
        print("2. Breadth-First Search (BFS)")
        choice = int(input("Enter your choice (1 or 2): "))

        match choice:
            case 1:
                print("Solving using DFS...")
                dfs_solution = dfs(initial_state, goal_state)
                if dfs_solution:
                    display_path(dfs_solution)
                else:
                    print("No solution found using DFS.")

            case 2:
                print("Solving using BFS...")
                bfs_solution = bfs(initial_state, goal_state)
                if bfs_solution:
                    display_path(bfs_solution)
                else:
                    print("No solution found using BFS.")

            case _:
                print("Invalid choice. Please select 1 or 2.")
```

```
Output: 1BM22CS290
Enter the initial state (3x3 grid, 0 for blank):
1 2 3
4 5 6
0 7 8
Enter the goal state (3x3 grid, 0 for blank):
1 2 3
4 5 6
7 8 0
Choose the method to solve the puzzle:
1. Depth-First Search (DFS)
2. Breadth-First Search (BFS)
Enter your choice (1 or 2): 2
Solving using BFS...
Step 0:
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]

Step 1:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

Step 2:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```