```
MAX_VISITED_DISPLAY = 10
NUM_INTERMEDIATE_STATES = 3
MAX_DEPTH_LIMIT = 50

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

def is_goal(state, goal_state):
    return state == goal_state

def find_zero(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def get_neighbors(state):
    neighbors = []
    x, y = find_zero(state)
    directions = [(1,0), (-1,0), (0,1), (0,-1)]
    for dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]
            neighbors.append(new_state)
    return neighbors

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

def dls(current_state, goal_state, depth_limit, path, visited, visited_states_display):
    """
    Depth-Limited Search helper for IDDFS.
```

```
    Returns:
        path if goal found else None
    """
    if len(path) - 1 > depth_limit:
        return None

    visited_states_display.append(current_state)
    if len(visited_states_display) <= MAX_VISITED_DISPLAY:
        print(f"Visited state #{len(visited_states_display)}:")
        print_state(current_state)

    if is_goal(current_state, goal_state):
        return path

    for neighbor in reversed(get_neighbors(current_state)):
        neighbor_tuple = tuple(tuple(row) for row in neighbor)
        if neighbor_tuple not in visited:
            visited.add(neighbor_tuple)
            result = dls(neighbor, goal_state, depth_limit, path + [neighbor], visited,
visited_states_display)
            if result is not None:
                return result
            # Backtrack visited set for other paths:
            visited.remove(neighbor_tuple)
    return None

def iddfs(start_state, goal_state, max_depth=MAX_DEPTH_LIMIT):
    """
    Iterative Deepening DFS:
    Tries DFS with increasing depth limits until goal found or max depth exceeded.
    """
    print("Starting Iterative Deepening DFS traversal...\n")

    for depth in range(max_depth + 1):
        print(f"Trying depth limit: {depth}")
        visited_states_display = []
        visited = set()
        visited.add(tuple(tuple(row) for row in start_state))
        path = dls(start_state, goal_state, depth, [start_state], visited, visited_states_display)
        if path is not None:
            print(f"\nGoal reached at depth {depth}!")
            print(f"Total visited states in last iteration: {len(visited_states_display)}")
            return path
        print(f"No solution found at depth {depth}\n")
```

```
    print(f"No solution found within max depth limit {max_depth}")
    return None

def read_state(name):
    print(f"Enter the {name} state, row by row (use space-separated numbers, 0 for empty):")
    state = []
    for _ in range(3):
        row = input().strip().split()
        if len(row) != 3:
            raise ValueError("Each row must have exactly 3 numbers.")
        row = list(map(int, row))
        state.append(row)
    return state

# --- Main Execution ---

initial_state = read_state("initial")
goal_state = read_state("goal")

if not (is_solvable(initial_state) == is_solvable(goal_state)):
    print("The puzzle is unsolvable.")
    exit()

solution_path = iddfs(initial_state, goal_state)

if solution_path:
    cost = len(solution_path) - 1
    print(f"\nSolution found with cost: {cost}\n")
    print("Solution path:")

    total_steps = len(solution_path) - 1  # number of moves
    print("Initial State:")
    print_state(solution_path[0])

    if total_steps > 1:
        step_indices = list(range(1, total_steps))
        if len(step_indices) > NUM_INTERMEDIATE_STATES:
            interval = len(step_indices) // (NUM_INTERMEDIATE_STATES + 1)
            selected_indices = [step_indices[i * interval] for i in range(1,
NUM_INTERMEDIATE_STATES + 1)]
        else:
            selected_indices = step_indices

        for idx in selected_indices:
```

```
            print(f"Intermediate State (Step {idx}):")
            print_state(solution_path[idx])

    print("Final State:")
    print_state(solution_path[-1])
else:
    print("No solution found")
```

```
Goal reached at depth 2!
Total visited states in last iteration: 7

Solution found with cost: 2

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

Intermediate State (Step 1):
1 2 3
4 5 0
6 7 8

Final State:
1 2 3
4 5 8
6 7 0
```

SUHAS BP (1BM23CS345)_