ITERATIVE DEEPENINF (DFS)

```
def get_neighbors(state):
    neighbors = []
    idx = state.index("0")
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # up, down, left, right
    x, y = divmod(idx, 3)

    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_idx = nx * 3 + ny
            state_list = list(state)
            state_list[idx], state_list[new_idx] = state_list[new_idx],
state_list[idx]
            neighbors.append("".join(state_list))
    return neighbors

def dfs_limit(start_state, goal_state, limit):
    stack = [(start_state, 0)] # Store state and depth
    visited = set()
    parent = {start_state: None}
    path = []

    while stack:
        current_state, depth = stack.pop()

        if current_state == goal_state:
             while current_state:
                path.append(current_state)
                current_state = parent[current_state]
             return path[::-1]

        if depth < limit and current_state not in visited:
            visited.add(current_state)
            neighbors = get_neighbors(current_state)
            neighbors.reverse() # Maintain consistent exploration order
            for neighbor in neighbors:
                if neighbor not in visited:
                    parent[neighbor] = current_state
                    stack.append((neighbor, depth + 1))
    return None

def iddfs(start_state, goal_state, max_depth):
    for limit in range(max_depth + 1):
        print(f"Searching with depth limit: {limit}")
        solution = dfs_limit(start_state, goal_state, limit)
        if solution:
            return solution
```

```
    return None


# Get input from the user row by row
print("1BM23CS333")
print("Enter the initial state (enter 3 digits per row, separated by
spaces, 0 for empty):")
initial_state_rows = []
for i in range(3):
    row = input(f"Row {i+1}: ").split()
    initial_state_rows.extend(row)
initial_state = "".join(initial_state_rows)

print("\nEnter the goal state (enter 3 digits per row, separated by
spaces, 0 for empty):")
goal_state_rows = []
for i in range(3):
    row = input(f"Row {i+1}: ").split()
    goal_state_rows.extend(row)
goal_state = "".join(goal_state_rows)

# Set a reasonable maximum depth for the search
max_depth = 50

solution = iddfs(initial_state, goal_state, max_depth)

if solution:
    print("\nIDDFS solution path:")
    for s in solution:
        print(s[:3])
        print(s[3:6])
        print(s[6:])
        print()
else:
    print(f"\nNo solution found within the maximum depth of
{max_depth}.")
```

OUTPUT:

```
1BM23CS333
Enter the initial state (enter 3 digits per row, separated by spaces, 0 for empty):
Row 1: 2 8 3
Row 2: 1 6 4
Row 3: 7 0 5

Enter the goal state (enter 3 digits per row, separated by spaces, 0 for empty):
Row 1: 1 2 3
Row 2: 8 0 4
Row 3: 7 6 5
Searching with depth limit: 0
Searching with depth limit: 1
Searching with depth limit: 2
Searching with depth limit: 3
Searching with depth limit: 4
Searching with depth limit: 5

IDDFS solution path:
283
164
705

283
104
765

203
184
765

023
184
765

123
084
765

123
804
765
```