Hill Climbing

Code:

```
class PuzzleState:
    def __init__(self, board, empty_tile_pos, moves=0, previous=None):
        self.board = board
        self.empty_tile_pos = empty_tile_pos
        self.moves = moves
        self.previous = previous

    def is_goal(self, goal):
        return self.board == goal

    def get_possible_moves(self):
        possible_moves = []
        x, y = self.empty_tile_pos
        directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Down, Up,
Right, Left

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [list(row) for row in self.board]  # Create a
copy
                new_board[x][y], new_board[new_x][new_y] =
new_board[new_x][new_y], new_board[x][y]
                possible_moves.append(PuzzleState(new_board, (new_x,
new_y), self.moves + 1, self))
        return possible_moves

def iddfs(initial_state, goal_state, depth_limit):
    def dls(state, depth):
        if state.is_goal(goal_state):
            return state
        if depth == 0:
            return None
```

```
        for move in state.get_possible_moves():
            result = dls(move, depth - 1)
            if result is not None:
                return result
        return None

    for depth in range(depth_limit):
        result = dls(initial_state, depth)
        if result is not None:
            return result
    return None

# Example initial and goal states
initial_board = [
    [1, 2, 3],
    [4, 0, 5],
    [7, 8, 6]
]
goal_board = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

# Initial state and goal state setup
initial_state = PuzzleState(initial_board, (1, 1))  # (1, 1) is the
position of the empty tile
goal_state = goal_board

# Set a depth limit
depth_limit = 1


# Run IDDFS
solution = iddfs(initial_state, goal_state, depth_limit)

# Function to print the solution path
def print_solution(solution):
    path = []
    while solution:
```

```
        path.append(solution.board)
        solution = solution.previous
    for step in reversed(path):
        for row in step:
            print(row)
        print()

if solution:
    print("Solution found:")
    print_solution(solution)
else:
    print("No solution found within the depth limit.")
```

Outputs:

```
PS C:\Users\Admin\Documents\Bramha> python -u "c:\Users\Ad
Enter the number of queens (N): 4
Iteration 0: Current state: [0, 2, 3, 1], Cost: 1
Local maximum reached at iteration 1. Restarting...
Iteration 0: Current state: [0, 3, 1, 1], Cost: 2
Iteration 1: Current state: [0, 3, 1, 2], Cost: 1
Local maximum reached at iteration 2. Restarting...
Iteration 0: Current state: [0, 0, 3, 1], Cost: 1
Iteration 1: Current state: [2, 0, 3, 1], Cost: 0
Solution found: [2, 0, 3, 1]
```