# Lab 3: HillClimbing_N_Queens

```
import random

def heuristic(state):
    """Calculate the number of conflicts between queens."""
    conflicts = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
                conflicts += 1
    return conflicts


def generate_neighbors(state):
    """Generate all neighboring states by swapping two queens."""
    neighbors = []
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            # Swap queens at positions i and j
            new_state = state.copy()
            new_state[i], new_state[j] = new_state[j], new_state[i]
            neighbors.append(new_state)
    return neighbors


def print_board(state):
    """Print the board configuration."""
    n = len(state)
    board = [["." for _ in range(n)] for _ in range(n)]
    for row in range(n):
```

```python
        board[row][state[row]] = "Q"
    for line in board:
        print(" ".join(line))
    print()


def get_user_input(n):
    """Get initial state from user input."""
    while True:
        try:
            input_state = input(f"Enter the initial positions for {n} queens (0 to {n-1}, space-separated): ")
            positions = list(map(int, input_state.split()))
            if len(positions) != n or any(p < 0 or p >= n for p in positions):
                raise ValueError
            return positions
        except ValueError:
            print("Invalid input. Please enter exactly {} numbers between 0 and {}.".format(n, n - 1))


def hill_climbing(n):
    """Perform the Hill-Climbing algorithm using the swapping technique."""
    # Get initial state from user
    current_state = get_user_input(n)
    current_cost = heuristic(current_state)

    print("Initial State:")
    print_board(current_state)
    print(f"Initial Heuristic (Conflicts): {current_cost}\n")

    while current_cost > 0:
        neighbors = generate_neighbors(current_state)
```

```
        next_state = None
        next_cost = current_cost  # Initialize with the current cost

        for neighbor in neighbors:
            cost = heuristic(neighbor)
            print(f"Evaluating Neighbor:")
            print_board(neighbor)
            print(f"Heuristic (Conflicts): {cost}")

            # Update the next state if a better (lower cost) neighbor is found
            if cost < next_cost:
                next_cost = cost
                next_state = neighbor

        if next_state is None:  # Local maximum reached
            print("Local maximum reached. No better neighbors found.")
            break  # Exit the loop

        # Move to the best neighbor found
        print("Moving to Next State:")
        current_state = next_state
        current_cost = next_cost
        print_board(current_state)
        print(f"Heuristic (Conflicts): {current_cost}\n")

    return current_state, current_cost  # Return the best found state and its cost

# Run the algorithm for the 4-queens problem
n = 4  # Change this value for different sizes of the board
```

solution, solution_cost = hill_climbing(n)

print("Best Solution Found:")

print_board(solution)

print(f"Final Heuristic (Conflicts): {solution_cost}")

## output:

```
Enter the initial positions for 4 queens (0 to 3, space-separated): 2
Invalid input. Please enter exactly 4 numbers between 0 and 3.
Enter the initial positions for 4 queens (0 to 3, space-separated): 1 2 0 3
Initial State:
. Q . .
. . Q .
Q . . .
. . . Q

Initial Heuristic (Conflicts): 1

Evaluating Neighbor:
. . Q .
. Q . .
Q . . .
. . . Q

Heuristic (Conflicts): 4
Evaluating Neighbor:
Q . . .
. . Q .
. Q . .
. . . Q

Heuristic (Conflicts): 2
Evaluating Neighbor:
. . . Q
. . Q .
Q . . .
. Q . .

Heuristic (Conflicts): 2
Evaluating Neighbor:
. Q . .
```

```
Q . . .
. . Q .
. . . Q

Heuristic (Conflicts): 2
Evaluating Neighbor:
. Q . .
. . . Q
Q . . .
. . Q .

Heuristic (Conflicts): 0
Evaluating Neighbor:
. Q . .
. . Q .
. . . Q
Q . . .

Heuristic (Conflicts): 4
Moving to Next State:
. Q . .
. . . Q
Q . . .
. . Q .

Heuristic (Conflicts): 0

Best Solution Found:
. Q . .
. . . Q
Q . . .
. . Q .

Final Heuristic (Conflicts): 0
```

In [ ]: