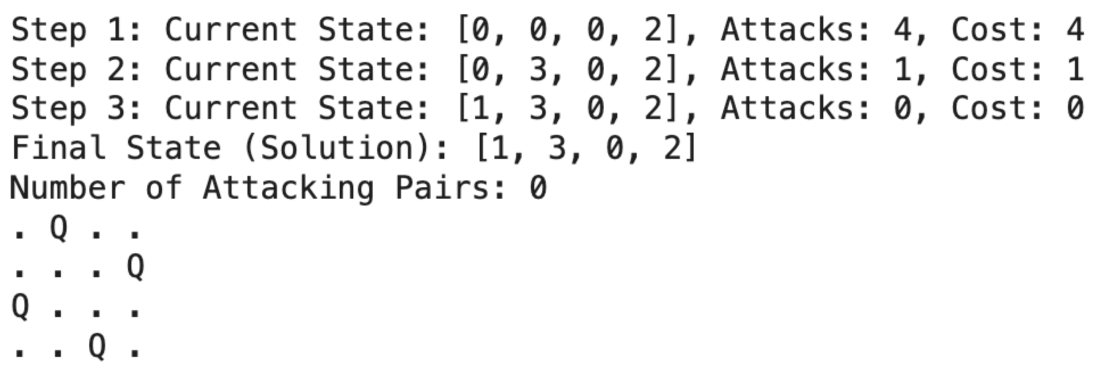
Lab - 4

Implement Hill Climbing Search Algorithm to solve N-Queens problem.

```python
import random

def get_attacking_pairs(state):
    """Calculates the number of attacking pairs of 
queens."""
    attacks = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:
                attacks += 1
            if abs(state[i] - state[j]) == abs(i - j):
                attacks += 1
    return attacks

def generate_successors(state):
    """Generates all possible successors by moving each 
queen to every other column in its row."""
    n = len(state)
    successors = []
    for row in range(n):
        for col in range(n):
            if col != state[row]:
                new_state[row] = col
                successors.append(new_state)
    return successors

def hill_climbing(n):
    """Hill climbing algorithm for n-queens problem."""
    current = [random.randint(0, n - 1) for _ in 
range(n)]
    steps = 0
    while True:
        current_attacks = get_attacking_pairs(current)
        successors = generate_successors(current)
        neighbor = min(successors, 
key=get_attacking_pairs)
        neighbor_attacks = get_attacking_pairs(neighbor)
        steps += 1
        print(f"Step {steps}: Current State: {current}, 
Attacks: {current_attacks}, Cost: {current_attacks}")
        if neighbor_attacks >= current_attacks:
            return current, current_attacks
        current = neighbor

def print_board(state):
    """Prints the board with queens placed."""
    n = len(state)
    board = [["." for _ in range(n)] for _ in range(n)]
    for row in range(n):
        board[row][state[row]] = "Q"
    for row in board:
        print(" ".join(row))
    print("\n")

n = 4
solution, attacks = hill_climbing(n)
print("Final State (Solution):", solution)
print("Number of Attacking Pairs:", attacks)
print_board(solution)
```

Output:



```
Step 1: Current State: [0, 0, 0, 2], Attacks: 4, Cost: 4
Step 2: Current State: [0, 3, 0, 2], Attacks: 1, Cost: 1
Step 3: Current State: [1, 3, 0, 2], Attacks: 0, Cost: 0
Final State (Solution): [1, 3, 0, 2]
Number of Attacking Pairs: 0
. Q . .
. . . Q
Q . . .
. . Q .
```