LAB 4
Hill Climbing

```python
import random
def calculate_cost(state):
    cost = 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):
                cost += 1
    return cost
def get_neighbors(state):
    neighbors = []
    n = len(state)
    for col in range(n):
        for row in range(n):
            if state[col] != row: # move queen in col to new row
                neighbor = state.copy()
                neighbor[col] = row
                neighbors.append(neighbor)
    return neighbors
def hill_climb(initial_state):
    current = initial_state
    current_cost = calculate_cost(current)

    print(f"Initial state: {current}, Cost = {current_cost}")

    while True:
        neighbors = get_neighbors(current)
        best_neighbor = None
        best_cost = current_cost
        for neighbor in neighbors:
            cost = calculate_cost(neighbor)
            if cost < best_cost:
                best_cost = cost
                best_neighbor = neighbor
        if best_neighbor is None:
            print(f"Final state: {current}, Cost = {current_cost}")
            return current, current_cost
        current, current_cost = best_neighbor, best_cost
        print(f"Move to: {current}, Cost = {current_cost}")
if __name__ == "__main__":
    n = int(input("Enter number of queens (N): "))
    print("Enter initial state as space-separated row positions for each column.")
    print("Example for N=4: '1 3 0 2' means queen at (0,1), (1,3), (2,0), (3,2).")
```

```
    current_cost = calculate_cost(current)

    print(f"Initial state: {current}, Cost = {current_cost}")

    while True:
        neighbors = get_neighbors(current)
        best_neighbor = None
        best_cost = current_cost
        for neighbor in neighbors:
            cost = calculate_cost(neighbor)
            if cost < best_cost:
                best_cost = cost
                best_neighbor = neighbor
        if best_neighbor is None:
            print(f"Final state: {current}, Cost = {current_cost}")
            return current, current_cost
        current, current_cost = best_neighbor, best_cost
        print(f"Move to: {current}, Cost = {current_cost}")
if __name__ == "__main__":
    n = int(input("Enter number of queens (N): "))
    print("Enter initial state as space-separated row positions for each column.")
    print("Example for N=4: '1 3 0 2' means queen at (0,1), (1,3), (2,0), (3,2).")

    initial_state = list(map(int, input("Initial state: ").split()))

    if len(initial_state) != n:
        print("Invalid input: Length of initial state must be N.")
    else:
        solution, cost = hill_climb(initial_state)

        if cost == 0:
            print(" Goal state reached!")
        else:
            print(" Stuck in local minimum.")
```

```
Enter number of queens (N):  4
Enter initial state as space-separated row positions for each column.
Example for N=4: '1 3 0 2' means queen at (0,1), (1,3), (2,0), (3,2).
Initial state:  0 3 0 3
Initial state: [0, 3, 0, 3], Cost = 3
Move to: [1, 3, 0, 3], Cost = 1
Move to: [1, 3, 0, 2], Cost = 0
Final state: [1, 3, 0, 2], Cost = 0
 Goal state reached!
```