WEEK-04

Implement **Hill Climbing Search** algorithm to **solve N-queens problem**

import random

def calculate\_cost(state):

    """Calculate the number of conflicts in the current 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):

    """Generate all possible neighbors by moving each queen in its column."""

    neighbors = []

    n = len(state)

    for col in range(n):

        for row in range(n):

            if state[col] != row:  # Move the queen in column `col` to a different row

                new\_state = list(state)

                new\_state[col] = row

                neighbors.append(new\_state)

    return neighbors

def hill\_climbing(n, max\_iterations=1000):

    """Perform hill climbing search to solve the N-Queens problem."""

    current\_state = [random.randint(0, n - 1) for \_ in range(n)]

    current\_cost = calculate\_cost(current\_state)

    for iteration in range(max\_iterations):

        if current\_cost == 0:  # Found a solution

            return current\_state

        neighbors = get\_neighbors(current\_state)

        neighbor\_costs = [(neighbor, calculate\_cost(neighbor)) for neighbor in neighbors]

        next\_state, next\_cost = min(neighbor\_costs, key=lambda x: x[1])

        if next\_cost >= current\_cost:  # No improvement found

            print(f"Local maximum reached at iteration {iteration}. Restarting...")

            return None  # Restart with a new random state

        current\_state, current\_cost = next\_state, next\_cost

        print(f"Iteration {iteration}: Current state: {current\_state}, Cost: {current\_cost}")

    print(f"Max iterations reached without finding a solution.")

    return None

# Get user-defined input for the number of queens

try:

    n = int(input("Enter the number of queens (N): "))

    if n <= 0:

        raise ValueError("N must be a positive integer.")

except ValueError as e:

    print(e)

    n = 4  # Default to 4 if input is invalid

solution = None

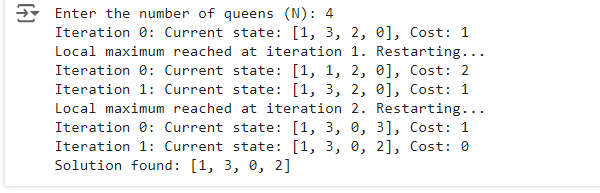
# Keep trying until a solution is found

while solution is None:

    solution = hill\_climbing(n)

print(f"Solution found: {solution}")

**OUTPUT:**

****