WEEK-05 HILL CLIMBING SEARCH ALG0RITHM

import random

def print\_board(board, n):

    """Prints the current state of the board."""

    for row in range(n):

        line = ""

        for col in range(n):

            if board[col] == row:

                line += " Q "

            else:

                line += " . "

        print(line)

    print()

def calculate\_conflicts(board, n):

    """Calculates the number of conflicts (attacks) between queens."""

    conflicts = 0

    for i in range(n):

        for j in range(i + 1, n):

            # Check if queens are in the same row or diagonal

            if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):

                conflicts += 1

    return conflicts

def get\_best\_neighbor(board, n):

    """

    Finds the best neighboring board with the fewest conflicts.

    Returns the best board and its conflict count.

    """

    current\_conflicts = calculate\_conflicts(board, n)

    best\_board = board[:]

    best\_conflicts = current\_conflicts

    neighbors = []

    for col in range(n):

        original\_row = board[col]

        for row in range(n):

            if row == original\_row:

                continue

            # Move queen to a new row and calculate conflicts

            board[col] = row

            new\_conflicts = calculate\_conflicts(board, n)

            neighbors.append((board[:], new\_conflicts))

        # Restore the original row before moving to the next column

        board[col] = original\_row

    # Sort neighbors by the number of conflicts (ascending)

    neighbors.sort(key=lambda x: x[1])

    if neighbors:

        best\_neighbor = neighbors[0]

        if best\_neighbor[1] < best\_conflicts:

            return best\_neighbor

    return board, current\_conflicts

def hill\_climbing\_with\_restarts(n, initial\_board, max\_restarts=100):

    """

    Performs Hill Climbing with random restarts to solve the N-Queens problem.

    Returns the final board configuration and its conflict count.

    """

    current\_board = initial\_board[:]

    current\_conflicts = calculate\_conflicts(current\_board, n)

    print("Initial board:")

    print\_board(current\_board, n)

    print(f"Initial conflicts: {current\_conflicts}\n")

    steps = 0

    restarts = 0

    while current\_conflicts > 0 and restarts < max\_restarts:

        new\_board, new\_conflicts = get\_best\_neighbor(current\_board, n)

        steps += 1

        print(f"Step {steps}:")

        print\_board(new\_board, n)

        print(f"Conflicts: {new\_conflicts}\n")

        if new\_conflicts < current\_conflicts:

            current\_board = new\_board

            current\_conflicts = new\_conflicts

        else:

            # If no better neighbor is found, perform a random restart

            restarts += 1

            print(f"Restarting... (Restart number {restarts})\n")

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

            current\_conflicts = calculate\_conflicts(current\_board, n)

            print("New initial board:")

            print\_board(current\_board, n)

            print(f"Conflicts: {current\_conflicts}\n")

    return current\_board, current\_conflicts

# Main function

def main():

    n = 4

    print("Enter the initial positions of queens (row numbers from 0 to 3 for each column):")

    initial\_board = []

    for i in range(n):

        while True:

            try:

                row = int(input(f"Column {i}: "))

                if 0 <= row < n:

                    initial\_board.append(row)

                    break

                else:

                    print(f"Please enter a number between 0 and {n-1}.")

            except ValueError:

                print("Invalid input. Please enter an integer.")

    solution, conflicts = hill\_climbing\_with\_restarts(n, initial\_board)

    print("Final solution:")

    print\_board(solution, n)

    if conflicts == 0:

        print("A solution was found with no conflicts!")

    else:

        print(f"No solution was found after {100} restarts. Final number of conflicts: {conflicts}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

print("Suvina A Shetty")

print("1BM22CS299")

OUTPUT