**Lab-04**

**Hill climbing (n queen)**

import random

def heuristic(board):

h = 0

n = len(board)

for i in range(n):

for j in range(i+1, n):

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

h += 1

return h

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

N = len(initial\_board)

board = [x-1 for x in initial\_board] # 0-based

h = heuristic(board)

restart\_count = 0

while h != 0 and restart\_count < max\_restarts:

steps = 0

while True:

best\_board = board[:]

best\_h = h

for col in range(N):

for row in range(N):

if row != board[col]:

neighbor = board[:]

neighbor[col] = row

h\_neighbor = heuristic(neighbor)

if h\_neighbor < best\_h:

best\_board = neighbor

best\_h = h\_neighbor

steps += 1

if best\_h >= h: # stuck

break

board = best\_board

h = best\_h

if h == 0:

break

if h == 0:

print(f"Solution found after {restart\_count} restarts and {steps} steps.")

break

# Random restart

board = [random.randint(0, N-1) for \_ in range(N)]

h = heuristic(board)

restart\_count += 1

return [x+1 for x in board], h

# User input

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

print(f"Enter the initial positions of {N} queens (row numbers 1 to {N}):")

initial\_board = list(map(int, input().split()))

solution, h\_val = hill\_climbing\_restart(initial\_board)

print("Final board:", solution)

print("Heuristic H =", h\_val)

