**Lab-05**

**Simulated annealing (n queen)**

import random

import math

def heuristic(board):

"""Count number of attacking pairs of queens."""

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 simulated\_annealing\_user\_initial(board, initial\_temp, cooling\_rate):

"""Simulated Annealing starting from user-provided initial board."""

N = len(board)

board = [x-1 for x in board] # convert to 0-based index

h = heuristic(board)

T = initial\_temp

steps = 0

while T > 0 and h != 0:

col = random.randint(0, N-1)

row = random.randint(0, N-1)

while row == board[col]:

row = random.randint(0, N-1)

neighbor = board[:]

neighbor[col] = row

h\_neighbor = heuristic(neighbor)

delta\_e = h\_neighbor - h

# Accept move

if delta\_e < 0:

board = neighbor

h = h\_neighbor

else:

if random.random() < math.exp(-delta\_e / T):

board = neighbor

h = h\_neighbor

T \*= cooling\_rate

steps += 1

if h == 0:

print(f"Solution found in {steps} steps.")

else:

print(f"Stopped after {steps} steps. Final H = {h} (may be stuck).")

return [x+1 for x in board], h # convert back to 1-based indexing

# 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()))

initial\_temp = float(input("Enter initial temperature: "))

cooling\_rate = float(input("Enter cooling rate (0 < rate < 1): "))

solution, h\_val = simulated\_annealing\_user\_initial(initial\_board, initial\_temp, cooling\_rate)

print("Final board:", solution)

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

