CODE:

import random

import math

def generate\_board(n):

"""Generate a random board configuration for N queens."""

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

def calculate\_cost(board):

"""Calculate the number of conflicts on the board."""

n = len(board)

conflicts = 0

# Count conflicts between pairs of queens

for i in range(n):

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

if board[i] == board[j]: # Same column

conflicts += 1

if abs(board[i] - board[j]) == abs(i - j): # Same diagonal

conflicts += 1

return conflicts

def get\_neighbors(board):

"""Generate all valid neighbors by moving a queen."""

neighbors = []

n = len(board)

for i in range(n):

for new\_row in range(n):

if new\_row != board[i]:

new\_board = board[:]

new\_board[i] = new\_row

neighbors.append(new\_board)

return neighbors

def print\_board(board):

"""Print the board as a matrix."""

n = len(board)

matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

matrix[i][board[i]] = 1

for row in matrix:

print(" ".join(str(cell) for cell in row))

def simulated\_annealing(n, initial\_temperature=100, cooling\_rate=0.95, max\_iterations=1000):

"""Solve N-Queens problem using Simulated Annealing."""

# Generate a random initial board configuration

current\_board = generate\_board(n)

current\_cost = calculate\_cost(current\_board)

temperature = initial\_temperature

iterations = 0

while current\_cost > 0 and iterations < max\_iterations:

neighbors = get\_neighbors(current\_board)

# Randomly select a neighbor

next\_board = random.choice(neighbors)

next\_cost = calculate\_cost(next\_board)

# Calculate the change in cost

delta\_cost = next\_cost - current\_cost

# Accept the new state with a certain probability

if delta\_cost < 0 or random.random() < math.exp(-delta\_cost / temperature):

current\_board = next\_board

current\_cost = next\_cost

# Reduce the temperature

temperature \*= cooling\_rate

iterations += 1

# Optionally, print the progress

if iterations % 100 == 0:

print(f"Iteration {iterations}: Cost = {current\_cost}")

if current\_cost == 0:

print("Solution found:")

print\_board(current\_board)

return current\_board

else:

print("Failed to find a solution within the maximum iterations.")

return None

# Set the size of the board (N x N)

n = 4 # Example for 4 queens

solution = simulated\_annealing(n)

if solution:

print("Final Board Configuration (as a matrix):")

print\_board(solution)

OUTPUT:

