TASK.8

# N-Queen Problem using Backtracking

def print\_solution(board):

"""Prints the chessboard configuration."""

for row in board:

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

print()

def is\_safe(board, row, col):

"""Checks if a queen can be placed at board[row][col]."""

N = len(board)

# Check left side of the current row

for i in range(col):

if board[row][i]:

return False

# Check upper diagonal on left side

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j]:

return False

# Check lower diagonal on left side

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j]:

return False

return True

def solve\_nq\_util(board, col):

"""Recursive utility function to solve the N-Queen problem."""

N = len(board)

# Base case: All queens placed

if col >= N:

print\_solution(board)

return True

res = False

for i in range(N):

if is\_safe(board, i, col):

board[i][col] = 1

res = solve\_nq\_util(board, col + 1) or res

board[i][col] = 0 # Backtrack

return res

def solve\_n\_queens(N):

"""Main function to solve N-Queen."""

board = [[0 for \_ in range(N)] for \_ in range(N)]

if not solve\_nq\_util(board, 0):

print("No solution exists")

return False

return True

# Example usage:

solve\_n\_queens(4)

OUTPUT:

