LP2 (AI) Lab Exp No.4

global N

N = 4

# ld is an array where its indices indicate row-col+N-1

ld = [0] \* 30

# rd is an array where its indices indicate row+col

rd = [0] \* 30

# Column array where its indices indicate column

cl = [0] \* 30

# A utility function to print solution

def printSolution(board):

for i in range(N):

for j in range(N):

print(" Q " if board[i][j] == 1 else " . ", end="")

print()

# A recursive utility function to solve N Queen problem

def solveNQUtil(board, col):

# Base case: If all queens are placed, return true

if col >= N:

return True

# Consider this column and try placing this queen in all rows one by one

for i in range(N):

# Check if the queen can be placed on board[i][col]

if (ld[i - col + N - 1] != 1 and rd[i + col] != 1) and cl[i] != 1:

# Place this queen in board[i][col]

board[i][col] = 1

ld[i - col + N - 1] = rd[i + col] = cl[i] = 1

# Recur to place the rest of the queens

if solveNQUtil(board, col + 1):

return True

# If placing the queen in board[i][col] doesn't lead to a solution, backtrack

board[i][col] = 0 # BACKTRACK

ld[i - col + N - 1] = rd[i + col] = cl[i] = 0

# If the queen cannot be placed in any row in this column col, return false

return False

# This function solves the N Queen problem using Backtracking.

# It mainly uses solveNQUtil() to solve the problem.

# It returns false if queens cannot be placed, otherwise,

# returns true and prints placement of queens in the form of 1s.

# Please note that there may be more than one solution;

# this function prints one of the feasible solutions.

def solveNQ():

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

if not solveNQUtil(board, 0):

print("Solution does not exist")

return False

printSolution(board)

return True

# Driver program to test above function

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

solveNQ()

OUTPUT: -

