18.Write the python program to solve B-Queen problem

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

# Check column

for i in range(row):

if board[i][col] == 1:

return False

# Check diagonal (top-left)

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

if board[i][j] == 1:

return False

# Check diagonal (top-right)

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

if board[i][j] == 1:

return False

return True

def solve\_b\_queen\_util(board, row, N, B, queens\_placed, solutions):

if queens\_placed == B:

# Found a valid solution

solution = [row[:] for row in board]

solutions.append(solution)

return

if row >= N:

return

for col in range(N):

if is\_safe(board, row, col, N):

board[row][col] = 1

solve\_b\_queen\_util(board, row + 1, N, B, queens\_placed + 1, solutions)

board[row][col] = 0

# Option to skip this row (no queen placed)

solve\_b\_queen\_util(board, row + 1, N, B, queens\_placed, solutions)

def solve\_b\_queen(N, B):

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

solutions = []

solve\_b\_queen\_util(board, 0, N, B, 0, solutions)

return solutions

def print\_board(board):

for row in board:

print(" ".join("Q" if col else "." for col in row))

print()

# Example usage

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

N = 5 # Board size

B = 3 # Number of queens

all\_solutions = solve\_b\_queen(N, B)

print(f"Total solutions for {B}-Queens on {N}x{N} board: {len(all\_solutions)}")

for idx, sol in enumerate(all\_solutions):

print(f"Solution #{idx + 1}")

print\_board(sol)

OUTPUT:

