### **AIM**

To solve the **8 Queens problem** using the **backtracking algorithm** in Python and visualize the result on a chessboard-like grid. The objective is to place 8 queens on an 8×8 chessboard so that no two queens threaten each other, i.e., no two queens share the same row, column, or diagonal.

### **Procedure / Algorithm**

1. **Create an 8×8 board** represented as a 2D list.
2. **Use a backtracking algorithm** to place a queen in each column from left to right.
3. At each step, check if placing a queen is **safe**:  
   * No queen in the same row to the left.
   * No queen in the upper-left diagonal.
   * No queen in the lower-left diagonal.
4. If a safe position is found, place the queen and move to the next column.
5. If placing the queen in the current row/column fails, **backtrack** to the previous column and move the queen.
6. Print the board when a valid configuration is found.

Code:

def print\_board(board):

for row in board:

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

print("\n" + "-" \* 16 + "\n")

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

for i in range(col):

if board[row][i]:

return False

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

if board[i][j]:

return False

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

if board[i][j]:

return False

return True

def solve\_queens(board, col):

if col >= len(board):

return True

for i in range(len(board)):

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

board[i][col] = 1

if solve\_queens(board, col + 1):

return True

board[i][col] = 0

return False

def solve\_8\_queens():

N = 8

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

if not solve\_queens(board, 0):

print("Solution does not exist.")

return

print("8 Queens Board:")

print\_board(board)

solve\_8\_queens()

Output:

8 Queens Board:

. Q . . . . . .

. . . . Q . . .

. . . . . . . Q

. . . Q . . . .

Q . . . . . . .

. . . . . Q . .

. . Q . . . . .

. . . . . . Q .