def solve\_n\_queens(n):

  """Solves the N-Queens problem using Branch and Bound and Backtracking.

  Args:

    n: The number of queens.

  Returns:

    A list of lists, where each inner list represents a row of the chessboard,

    and each element in the inner list represents the column of the queen in that

    row.

  """

  # Create a board to represent the chessboard.

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

  # Use backtracking to solve the problem.

  def backtrack(row):

    """Recursively solves the N-Queens problem using backtracking.

    Args:

      row: The current row of the chessboard.

    Returns:

      True if a solution was found, False otherwise.

    """

    # If we have reached the last row, we have found a solution.

    if row == n:

      return True

    # Try placing a queen in each column of the current row.

    for col in range(n):

      # If placing a queen in the current column is not valid, backtrack.

      if not is\_valid\_placement(board, row, col):

        continue

      # Place a queen in the current column.

      board[row][col] = 1

      # Recursively solve the problem for the next row.

      if backtrack(row + 1):

        return True

      # If no solution was found for the next row, backtrack.

      board[row][col] = 0

    # If no solution was found for the current row, backtrack.

    return False

  # Solve the problem using backtracking.

  if not backtrack(0):

    return None

  # Return the solution.

  return board

def is\_valid\_placement(board, row, col):

  """Checks if placing a queen in the given row and column is valid.

  Args:

    board: The chessboard.

    row: The row to place the queen in.

    col: The column to place the queen in.

  Returns:

    True if placing a queen in the given row and column is valid, False

    otherwise.

  """

  # Check if there is a queen in the same row.

  for i in range(col):

    if board[row][i] == 1:

      return False

  # Check if there is a queen in the same column.

  for i in range(row):

    if board[i][col] == 1:

      return False

  # Check if there is a queen in the same diagonal.

  for i in range(row):

    for j in range(col):

      if board[i][j] == 1 and (i - j == row - col or i + j == row + col):

        return False

  # If there is no queen in the same row, column, or diagonal, the placement

  # is valid.

  return True

# Solve the N-Queens problem for n = 4.

solution = solve\_n\_queens(4)

# Print the solution.

for row in solution:

  print(row)