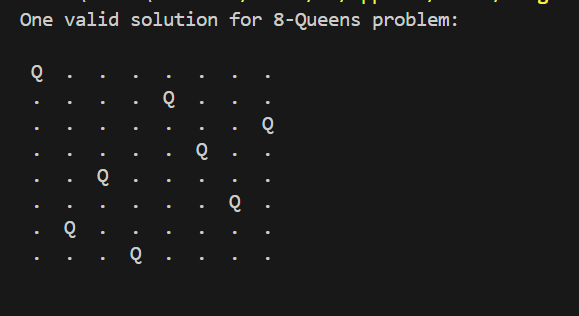
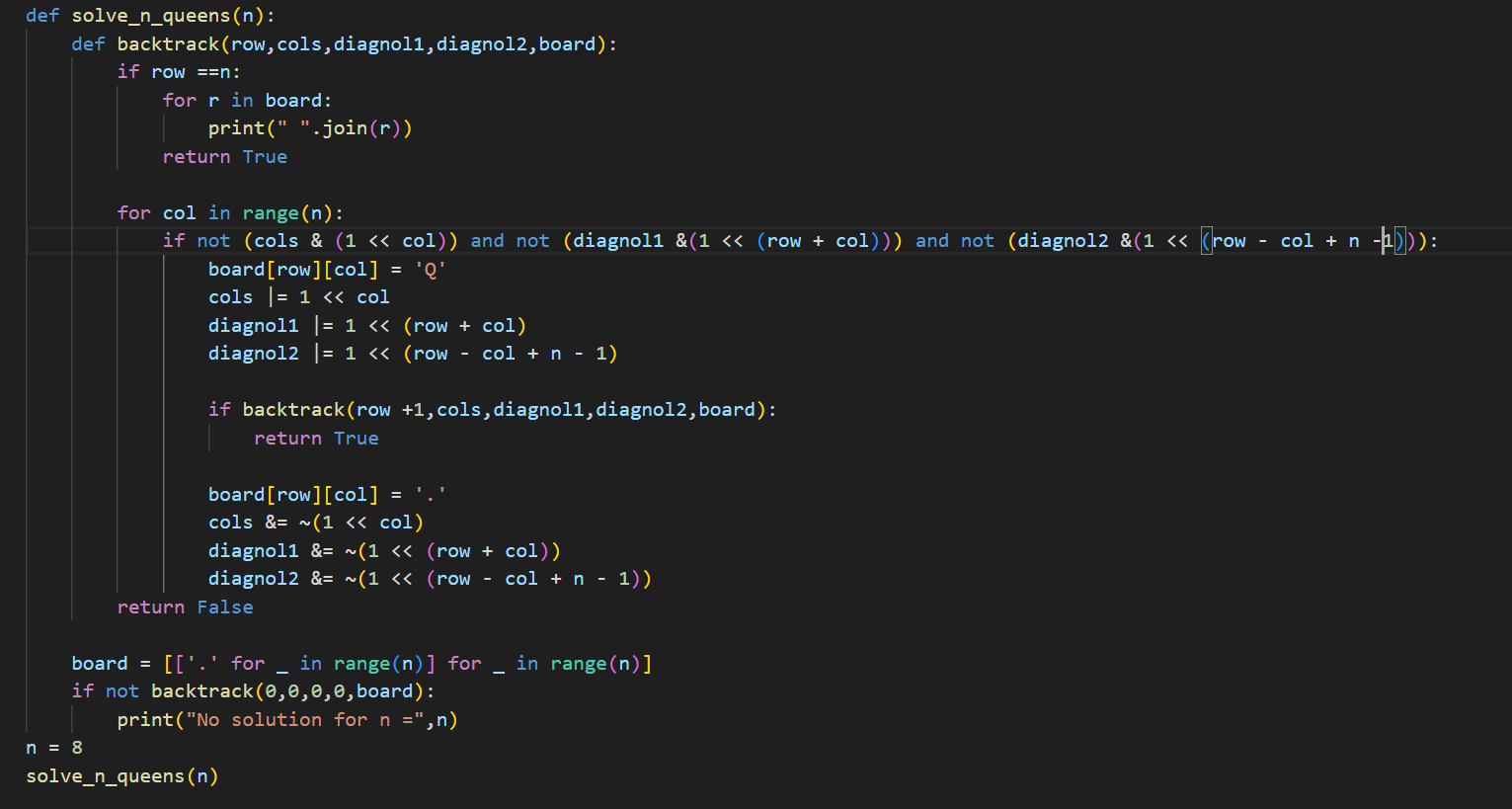
**Lab 4**

**Task: N-Queens Problem (Dynamic)**



**How the Code Works**

The solve\_n\_queens function initializes an **N x N chessboard** filled with '.' to represent empty spaces and calls the backtrack function to start solving the problem. The backtrack function places queens **row by row** while ensuring no conflicts occur. It takes several parameters: row (current row), cols (bitmask for occupied columns), diagnol1 (bitmask for occupied diagonals from top-left to bottom-right), diagnol2 (bitmask for occupied anti-diagonals from top-right to bottom-left), and board (current chessboard state). If row == n, it means all queens have been successfully placed without conflicts, so the board is printed, and True is returned. Otherwise, for each column in the current row, the function checks if placing a queen is **safe** using bitmask operations: cols & (1 << col) for columns, diag1 & (1 << (row + col)) for diagonals, and diag2 & (1 << (row - col + n - 1)) for anti-diagonals. If a position is safe, the queen is placed (board[row][col] = 'Q'), and bitmasks are updated before recursively moving to the next row (row + 1). If a valid solution is found, True propagates up the call stack. Otherwise, if no solution exists in the current path, the queen is removed (board[row][col] = '.'), bitmasks are reset, and the function continues to the next column. If no valid placement is found for the current row, the function **backtracks** to the previous row to explore other possibilities.

**Why the Code Works**

The code efficiently solves the N-Queens problem using **bitmasking and backtracking**. Instead of using loops to check for conflicts, it leverages **bitwise operations**: cols & (1 << col) checks if a column is occupied, while diag1 & (1 << (row + col)) and diag2 & (1 << (row - col + n - 1)) verify conflicts on main and anti-diagonals, respectively. This optimization reduces the conflict-checking time complexity from **O(N) to O(1)**, significantly improving performance. The algorithm explores all possible queen placements row by row, backtracking when conflicts arise to systematically explore all valid configurations. To optimize further, the function **te**rminates immediately upon finding the first valid solution, preventing unnecessary computations and ensuring only one correct board configuration is printed.