

The Superior University

|  |  |  |
| --- | --- | --- |
| Name: Muhammad Bilal Sajid | Roll No: 018 | Course: PAI Lab |
| Semester: 4th | Section: BSAI 4A | Department: |
| Submitted To: | Total Marks: | Date: |

**Lab 4**

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

**N-Queens Problem Solver**

**Code Summary:**

This Python program implements a solution for the **N-Queens problem**, a classic backtracking problem where N queens must be placed on an N x N chessboard such that no two queens attack each other.

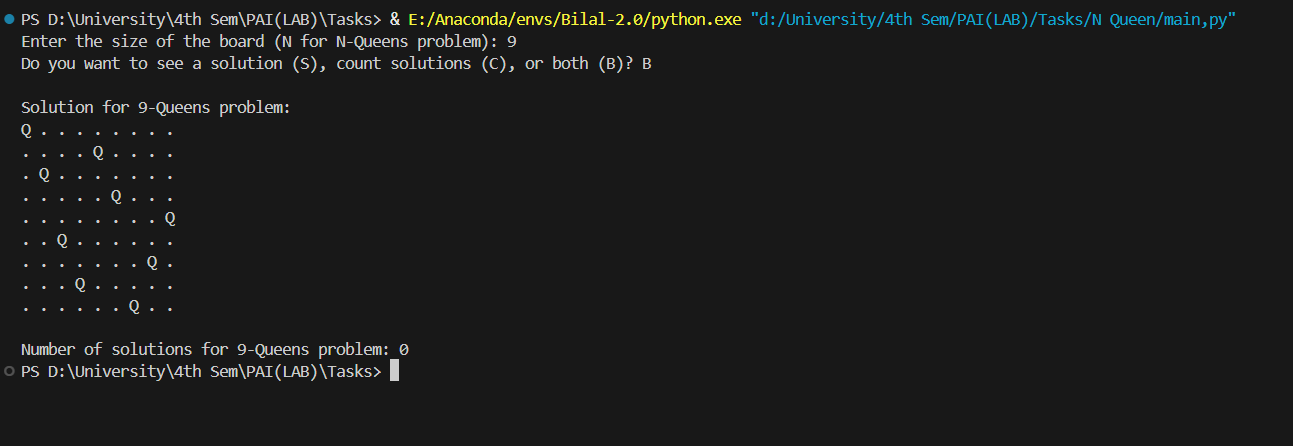
**How the Code Works:**

1. **Solving the N-Queens Problem:**
   * The function solve\_n\_queens(n) finds a valid configuration of N queens using **backtracking**.
   * The is\_safe(board, row, col) function checks if placing a queen at a given position is valid by ensuring there are no other queens in the same column, left diagonal, or right diagonal.
   * The backtrack(board, col) function recursively attempts to place queens in each column, backtracking when a conflict occurs.
   * If a valid configuration is found, the solution is returned as a list where each index represents a row, and the value at that index represents the column position of the queen.
2. **Printing the Board:**
   * print\_board(solution, n) visually represents the chessboard with Q for queens and . for empty spaces.
   * If no solution exists, it prints an appropriate message.
3. **Counting All Possible Solutions:**
   * The count\_solutions(n) function counts the number of distinct valid solutions using **backtracking**.
   * Instead of using a 2D board, this function uses a 1D array queens, where the index represents a row and the value represents the column position.
   * The function is\_safe(queens, row, col) checks if a queen can be safely placed in a given position.
   * The backtrack(queens, row) function iterates over all columns, trying to place queens and recursively counting solutions.
4. **User Interaction:**
   * The program asks the user for the board size N.
   * It provides options to either **display a solution**, **count the total solutions**, or **both**.
   * Based on user input, the program calls solve\_n\_queens() and/or count\_solutions() accordingly.

**Why this Approach?**

* **Backtracking** ensures all possibilities are explored efficiently while pruning invalid paths early.
* The counting approach optimizes space by using a **1D list instead of a full chessboard**.
* The solution visualization helps in understanding queen placements.

**Example Output Screenshot:**

****