

**Name: Hafiz Haider Ali**

**Roll No: BSAI-130-4C**

**Date: 27 Feb 2025**

**Subject: PAI LAB**

# N-Queens Problem Solver

This Python program solves the classic N-Queens problem using a backtracking algorithm with dynamic visualization. The goal is to place N queens on an NxN chessboard such that no two queens threaten each other.

## Workflow Overview

The program follows these key steps:

### 1. Importing Necessary Libraries

* os: For clearing the console screen
* time: For adding delays in visualization

### 2. Defining Core Functions

#### is\_safe(board, row, col, n)

* **Purpose**: Checks if a queen can be safely placed at position (row, col)
* **Process**:
  + Checks the row to the left
  + Checks upper-left diagonal
  + Checks lower-left diagonal
* **Returns**: True if safe, False otherwise

#### print\_board\_dynamic(board, n, delay=0.2)

* **Purpose**: Displays the current board state with animation
* **Process**:
  + Clears screen using clear\_screen()
  + Prints board with 'Q' for queens and '.' for empty spaces
  + Adds delay for visual effect

#### solve\_n\_queens\_util(board, col, n, solutions, visualize)

* **Purpose**: Recursive function to solve the problem
* **Process**:
  + Base case: If all columns filled, saves solution
  + Tries each row in current column
  + Places queen if safe, recursively solves for next column
  + Backtracks if needed
* **Visualization**: Shows progress if enabled

#### solve\_n\_queens(n, visualize)

* **Purpose**: Main solver function
* **Process**:
  + Initializes NxN board
  + Calls utility function
  + Returns solutions

### 3. Loading the Problem

* User inputs the value of N (board size)
* No external data files needed

### 4. board Initialization

* Creates an NxN matrix filled with zeros
* Example for N=4:

### 5. Visualizing Progress

* print\_board\_dynamic() shows:
  + Initial empty board
  + Queens being placed (1s)
  + Backtracking (removing 1s, replacing with 0s)
* Example output for N=4 during solving:

### 6. Splitting the Problem

* **Features**: Board positions (rows and columns)
* **Target**: Valid placement of N queens
* No explicit train/test split as this is a constraint satisfaction problem

### 7. Solving Strategy

* **Algorithm**: Backtracking
* Tries placing queens column by column
* Backtracks when a placement fails
* Stores all valid solutions

### 8. Building the Solution Pipeline

* **Steps**:
  1. Check safety of position
  2. Place queen if safe
  3. Move to next column
  4. Backtrack if no solution found
* **Visualization**: Optional real-time display

### 9. Running the Solver

* **Input**: User-provided N
* **Process**:
  + Validates input (N ≥ 1, handles N=2,3 cases)
  + Shows solving process
  + User can interrupt with Ctrl+C
* **Output**: Number of solutions and one example

### 10. Results

* **For N=1**: 1 solution
* **For N=2,3**: No solutions possible
* **For N=4**: 2 solutions
* **Example Output (N=4)**:

### 11. Accuracy

* **Correctness**: Always finds all valid solutions
* **Completeness**: Explores all possibilities
* **Time Complexity**: O(N!) due to backtracking

### 12. User Interaction

* Prompts for N value
* Option to try multiple N values
* Ends with "Thanks for playing!"