****

**PROGRAMMING FOR AI (LAB)**

**Lab Task No 04**

**Submitted To :- Sir Rasikh Ali**

**Submitted By :- Aiman Ijaz**

**Roll No :- SU92-BSAIM-F23-010**

**Department :- Software Engineering**

**Section :- BSAI-4A**

**N-Queens Problem Dynamic**

**Introduction**

The **N-Queens problem** is a classic **constraint satisfaction problem** where N queens must be placed on an N × N chessboard so that no two queens **attack each other**. This means that no two queens can be placed in the **same row, column, or diagonal**. The provided solution uses **backtracking** to find a valid arrangement of queens.

**How the Code Works**

**1. Board Initialization (initialize(n))**

* The chessboard is **represented as a dictionary** (board) containing:
  + 'queen': Stores the column index for each row where a queen is placed.
  + 'row': Keeps track of which rows are occupied.
  + 'col': Keeps track of which columns are occupied.
  + 'nwtose': Represents the **Northwest-to-Southeast** diagonal (\).
  + 'swtone': Represents the **Southwest-to-Northeast** diagonal (/).
* Initially:
  + All rows and columns are **unoccupied**.
  + The queen positions are set to **-1** (indicating no queen is placed).

**2. Checking if a Position is Safe (free(i, j))**

* This function checks if placing a queen at row i, column j is **valid**:
  + The row i is **not occupied**.
  + The column j is **not occupied**.
  + The **diagonal** (\) nwtose[j - i] is **not occupied**.
  + The **diagonal** (/) swtone[j + i] is **not occupied**.

**3. Placing and Removing a Queen (addqueen(i, j) and undoqueen(i, j))**

* If a queen is placed at (i, j), the function **updates**:
  + board['queen'][i] = j → Stores the queen’s position.
  + board['row'][i] = 1 → Marks the row as occupied.
  + board['col'][j] = 1 → Marks the column as occupied.
  + board['nwtose'][j - i] = 1 → Marks diagonal (\) as occupied.
  + board['swtone'][j + i] = 1 → Marks diagonal (/) as occupied.
* If backtracking is required, undoqueen(i, j) **removes** the queen by resetting these values.

**4. Recursive Backtracking (placequeen(i))**

* This function **tries to place a queen** in row i:
  1. It iterates through each column j and **checks if it's safe**.
  2. If safe, it **places the queen** and recursively **calls placequeen(i+1)**.
  3. If all queens are placed (i == n-1), the function **returns True (success)**.
  4. If no valid placement exists, it **removes the last placed queen** and backtracks.

**5. Printing the Board (printboard())**

* Once a valid solution is found, it prints the chessboard with **‘Q’** representing the queens.

**Output:**

****