**2. Problem Understanding**

* **Input**: An integer N, the size of the chessboard (N×N), representing the number of queens to place.
* **Output**: All possible ways to place the N queens on the board such that no two queens threaten each other. Each solution is represented as a configuration of queens on the board.

The queens must be placed on the chessboard such that:

* No two queens share the same row.
* No two queens share the same column.
* No two queens share the same diagonal.

**3. Solution Explanation**

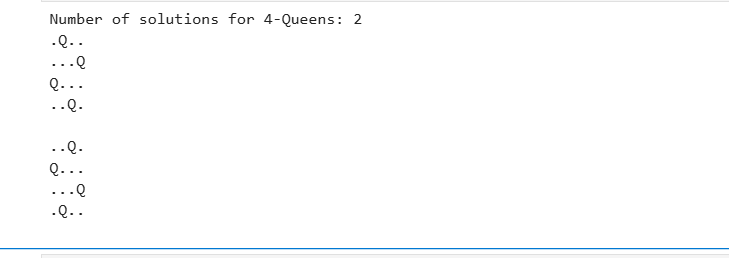
The solution to the N-Queens problem is implemented using **backtracking**. Below is the explanation of the approach and the code.

**3.1. Backtracking Approach**

The algorithm uses **backtracking** to explore all possible configurations of placing queens on the board. Here's a step-by-step breakdown of the approach:

1. **Recursive Exploration**:
   * Start by placing a queen in the first row, then move to the next row and place another queen in one of its columns.
   * Repeat this process for each row, trying to place a queen in each column.
2. **Safety Check**:
   * After placing each queen, we must check if it conflicts with any previously placed queens. This is done using a helper function that checks:
     + Column conflicts.
     + Diagonal conflicts.
   * If a queen cannot be placed safely in a column, we backtrack by removing the queen from the previous row and trying a different column.
3. **Base Case**:
   * If a queen is placed in all N rows (i.e., we've placed all N queens successfully), we record the configuration as a valid solution.
4. **Backtracking**:
   * If at any point, no column is found where a queen can be placed in a given row, the algorithm backtracks to the previous row and tries the next possible column for the queen there.

**3.2. Code Explanation**

****

* The recursive function that attempts to place queens row by row.
  + **Base case**: If all rows have been filled with queens (row == n), the current configuration is added to the solutions list.
  + It tries placing a queen in each column for the current row and recursively tries to place queens in the next row.
  + **Backtracking**: If placing a queen leads to a conflict, the queen is removed (board[row] = -1) and a different column is tried.

**3.2.4. print\_solutions(solutions)**

* This function takes the list of solutions and prints each one in a readable format. Each solution is represented by a list where each index corresponds to a row, and the value at that index is the column where the queen is placed.

**4. Explanation of the Code Logic**

The code explores all possible ways to place queens on the board by considering each row and column, backtracking when it encounters a conflict. The steps taken by the algorithm are:

1. **Initialization**: We initialize an empty board and a list to store solutions.
2. **Recursion**: We recursively try to place queens in each row, starting from the first row.
3. **Safety Check**: Before placing a queen, the algorithm checks if placing it at the current position will lead to any conflicts.
4. **Backtracking**: If a solution leads to an invalid state, we backtrack and try a different placement for previous queens.
5. **Solution Found**: Once a valid configuration is found, it is added to the list of solutions.

This backtracking method ensures that all possible configurations are explored while eliminating invalid ones early.