N Queen Problem II LeetCode

The **n-queens** puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return *the number of distinct solutions to the **n-queens puzzle***.

Example: N = 4

Output: The Distinct number of solutions of N-Queen Puzzle: 2

Approach 1: Function to calculate the total number of distinct solutions for the N-Queens problem Backtracking approach

Function Purpose:

Calculate the total number of distinct solutions for the N-Queens problem using the backtracking approach.

Explanation:

- **isPossible Function:**
    - Checks if placing a Queen at the specified position is feasible, considering the row, column, and diagonals.
- **solve Function:**
    - Recursive backtracking function to explore all possible placements of Queens on the chessboard.
    - Increments the count when a valid configuration is found.
- **totalNQueens Function:**
    - Initializes an empty chessboard and starts solving from the first column using the **solve** function.
    - Returns the total count of distinct solutions.

Time Complexity:

- **Backtracking per Queen Placement: O(N!), where N is the size of the chessboard (number of queens).**

Space Complexity:

- **Chessboard Storage: O(N^2), where N is the size of the chessboard.**

Approach 2: Function to calculate the total number of distinct solutions for the N-Queens problem using optimized approach

Function Purpose:

Calculate the total number of distinct solutions for the N-Queens problem using an optimized backtracking approach.

Explanation:

- **isPossible Function:**
    - Checks if placing a Queen at the specified position is feasible using hash maps to track occupied rows and diagonals.
- **setMapValues Function:**
    - Sets values in hash maps when placing a Queen.
- **resetMapValues Function:**
    - Resets values in hash maps during backtracking.
- **nQueensHelper Function:**
    - Recursive backtracking function to explore all possible placements of Queens on the chessboard using hash maps.
    - Increments the count when a valid configuration is found.
- **totalNQueensOptimized Function:**
    - Initializes an empty chessboard and starts solving from the first column using the **nQueensHelper** function.
    - Returns the total count of distinct solutions.

Time Complexity:

- **Backtracking per Queen Placement: O(N!), where N is the size of the chessboard (number of queens).**

Space Complexity:

- **Chessboard Storage: O(N^2), where N is the size of the chessboard.**
- **Hash Map Storage: O(N).**

Conclusion:

- Both approaches calculate the total number of distinct solutions for the N-Queens problem using backtracking.

- The optimized approach reduces redundant checks using hash maps for row and diagonal occupancy.
- The time complexity remains exponential due to the nature of the problem.