# Name: Hashim Nadeem

**RollNo: SU92-BDSFM-F25-001**

# N-Queens Problem using Dynamic Programming (Python)

## Introduction

The N-Queens Problem is a classic combinatorial problem in computer science and artificial intelligence. The goal is to place N queens on an N×N chessboard such that no two queens threaten each other — meaning that no two queens share the same row, column, or diagonal.

## Objective

To implement an efficient dynamic programming (DP) solution in Python that counts all possible valid arrangements of N queens using bitmasking and memoization.

## Problem Description

Given a chessboard of size N×N, place N queens on the board so that no two queens attack each other. The problem can be solved using recursive search or backtracking, but this implementation uses a more optimized approach based on Dynamic Programming with bitmasks and memoization.

## Algorithm Overview

1. Each state of the board is represented by three bitmasks:  
 - Columns already occupied by queens.  
 - Main diagonals occupied.  
 - Anti-diagonals occupied.  
2. The recursion proceeds row by row, choosing valid positions for queens based on free columns.  
3. Memoization is used to store and reuse subproblem results to avoid redundant calculations.  
4. The algorithm continues until all queens are placed, and counts or generates valid boards.

## Functions Description

1. count\_nqueens(n): Returns the total number of valid solutions for N queens.

2. generate\_solutions(n, limit): Generates up to 'limit' valid board configurations as text grids.

## Example Execution

For example, when N = 8 (the 8-Queens Problem), the algorithm finds all 92 possible solutions. Using generate\_solutions(8, limit=2), it displays the first two valid configurations.

## Sample Output

Total solutions for N=8: 92  
  
Solution 1:  
Q.......  
....Q...  
.......Q  
.....Q..  
..Q.....  
......Q.  
.Q......  
...Q....  
  
Solution 2:  
Q.......  
.....Q..  
.......Q  
..Q.....  
......Q.  
...Q....  
.Q......  
....Q...

## Advantages of Dynamic Programming Approach

* Uses bitmasks for compact and fast representation of state.
* Avoids recomputation with memoization, improving efficiency.
* Highly scalable for larger N compared to naive recursion.
* Easily adaptable for both counting and generating solutions.

## Conclusion

The dynamic programming solution to the N-Queens Problem demonstrates the power of state compression and memoization. By using bitmasks, the algorithm can efficiently explore and count valid configurations for even large N values. This implementation also allows generating a subset of valid boards for visualization.