In [1]:

```
# 1. Helper functions to check is the postions are safe

def is_safe(board, row, col, n):
    # Check the left side of the current row
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper-left diagonal
    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1

    # Check lower-left diagonal
    i, j = row, col
    while i < n and j >= 0:
        if board[i][j] == 1:
            return False
        i += 1
        j -= 1

    return True
```

In [2]:

```
# 2. Backtracking function for CSP Solver

def solve_n_queens_util(board, col, n, solutions):
    if col == n:
        solutions.append(["".join("Q" if cell else "." for cell in row) for row
        return

    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = 1
            solve_n_queens_util(board, col + 1, n, solutions)
            board[i][col] = 0  # Backtrack
```

In [3]:

```
# 3. Main Function to solve the CSP Problem

def solve_n_queens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    solutions = []
    solve_n_queens_util(board, 0, n, solutions)
    return solutions
```

In [4]:

```
# 4. Function to print the solution

def print_solutions(solutions):
    print(f"\nTotal Solutions: {len(solutions)}")
    for idx, solution in enumerate(solutions, 1):
        print(f"\nSolution {idx}:")
        for row in solution:
            print(row)
```

In [5]:

```
# 5. Solving the N - Queen Problem

N = int(input("Enter the value of N for the N-Queens problem: "))
solutions = solve_n_queens(N)
print_solutions(solutions)
```

```
Enter the value of N for the N-Queens problem: 4

Total Solutions: 2

Solution 1:
..Q.
Q...
...Q
.Q..

Solution 2:
.Q..
...Q
Q...
..Q.
```