N_QUEEN PROBLEM REPORT

Name-RAVI KISHAN

UNIVERSITY ROLL – 202401100400154

DATE – 10 March 2025

Cseaiml_c

KIET GROUP OF INSTITUTIONS

Introduction :-

The N-Queen problem is a classic combinatorial problem in which N queens must be placed on an N×N chessboard so that no two queens threaten each other. This means that no two queens can be placed in the same row, column, or diagonal. This report focuses on solving the N-Queen problem for a 4×4 chessboard using a backtracking algorithm.

Methodology : -

The solution to the N-Queen problem is implemented using a backtracking algorithm. The algorithm places queens one by one in different columns, ensuring that each placement does not violate the rules of the problem. If a placement leads to a conflict, the algorithm backtracks and tries the next possible placement. The process continues until all queens are successfully placed on the board.

Steps:

1. Start from the first row and place a queen in a safe column.
2. Move to the next row and repeat the process.
3. If a row has no valid column, backtrack to the previous row and change the queen's position.
4. Repeat until all queens are placed or all configurations are exhausted

CODE :-

```
def solve_n_queens(n):
    """
    Solves the N-Queens problem and returns all possible solutions.
    Each solution is represented as a list of strings, where 'Q' marks a queen and '.' marks an empty space.
    """


    def backtrack(row, cols, diag1, diag2, board):
        """
            Recursive backtracking function to place queens row by row.
```

```
    Args:
    - row: The current row being processed.
    - cols: A set tracking occupied columns.
    - diag1: A set tracking occupied major diagonals (row - col).
    - diag2: A set tracking occupied minor diagonals (row + col).
    - board: The current board state.
    """

    if row == n:  # Base case: all rows are filled, valid solution found
        solutions.append(["".join(r) for r in board])  # Store board configuration
        return

    for col in range(n):  # Try placing a queen in each column
            if col in cols or (row - col) in diag1 or (row + col) in diag2:
            continue  # Skip if this position is under attack

        # Place queen
        board[row][col] = 'Q'

            # Recurse to the next row with updated constraints
```

```
                backtrack(row + 1, cols | {col}, diag1 | {row - col}, diag2 | {row + col}, board)

            # Remove queen (backtrack)
            board[row][col] = '.'

    # Initialize variables
    solutions = []  # Stores all valid solutions
    board = [["."] * n for _ in range(n)]  # Create an empty NxN chessboard

    # Start backtracking from the first row
    backtrack(0, set(), set(), set(), board)

    return solutions  # Return all solutions

# Example usage
for sol in solve_n_queens(4):
    print("\n".join(sol), "\n")
```

CODE OUTPUT SCREENSHOT :-

```
# Example usage
for sol in solve_n_queens(4):
    print("\n".join(sol), "\n")
```

```
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..
```