6.1 N-QUEENS PROBLEM VISUALIZATION

Question:

Discuss the importance of visualizing the solutions of the N-Queens Problem to understand the placement of queens better. Use a graphical representation to show how queens are placed on the board for different values of N. Explain how visual tools can help in debugging the algorithm and gaining insights into the problem's complexity. Provide examples of visual representations for N = 4, N = 5, and N = 8, showing different valid solutions.

AIM

To implement the N-Queens Problem in Python and visualize the placement of queens on the chessboard for different values of N.

ALGORITHM

1. Place queens row by row on the chessboard.
2. At each row, check all columns and place the queen in a safe column (no two queens attack each other).
3. Mark attacked columns and diagonals.
4. Use backtracking: If a placement leads to a dead-end, backtrack and try another column.
5. Stop when N queens are placed successfully.
6. Display the board with queens (Q) and empty cells (.).

PROGRAM

```
def solve_n_queens(n):
    def is_safe(row, col, board):
        for r in range(row):
            if board[r] == col or abs(board[r] - col) == abs(r - row):
                return False
        return True

    def backtrack(row, board, solutions):
        if row == n:
            solutions.append(board[:])
            return
        for col in range(n):
            if is_safe(row, col, board):
                board[row] = col
                backtrack(row + 1, board, solutions)

    solutions = []
    backtrack(0, [-1]*n, solutions)
    return solutions

def print_board(board):
    n = len(board)
    for row in board:
        line = ['.'] * n
        line[row] = 'Q'
        print(' '.join(line))
    print()

n = int(input("Enter value of N for N-Queens: "))
solutions = solve_n_queens(n)
print(f"Showing up to 3 valid solutions for N = {n}:")
for sol in solutions[:3]:
    print_board(sol)
```

Input:

Enter value of N for N-Queens: 4

Output:

```
Enter value of N for N-Queens: 4
Showing up to 3 valid solutions for N = 4:
. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .

>>>
```

RESULT:

Thus, the program is successfully executed, and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: O(N!) in the worst case due to backtracking search.
- Space Complexity: O(N) for storing column and diagonal constraints.