

## **Exp-6.1**

### **Title:**

Visualization of the N-Queens Problem Solutions

### **Aim:**

To understand the placement of queens on an  $N \times N$  chessboard through graphical representations for different values of  $N$  (4, 5, and 8) and to demonstrate how visualization aids in debugging and gaining insights into the problem's complexity.

### **Algorithm**

1. Initialize an empty  $N \times N$  board with '.' representing empty cells.
2. Start placing queens row by row, beginning from the first row.
3. For the current row, iterate over each column to find a safe position.
4. Check if placing a queen at the current cell does not conflict with previously placed queens in:
  1. The same column
  2. The upper left diagonal
  3. The upper right diagonal
5. If safe, place the queen ('Q') at the current cell and move to the next row recursively.
6. If a conflict occurs or no valid placements are possible in a row, backtrack by removing the queen from the previous row and try different positions.
7. Repeat steps 3-6 until all queens are placed or all possible placements are exhausted.

**Input:**

$N = 4$

$N = 5$

**Output:**

.Q..

...Q

Q...

..Q.

Q....

..Q..

....Q

.Q...

...Q.

**Performance Analysis:**

**Time Complexity:**  $O(N!)$

**Space Complexity:**  $O(N^2)$

**Program output :**

```
def solveNQueens(N):  
    board = [['.' for _ in range(N)] for _ in range(N)]  
  
    def isSafe(row, col):  
        # Check column conflicts  
        for i in range(row):  
            if board[i][col] == 'Q':  
                return False  
        # Check upper left diagonal  
        i, j = row - 1, col - 1  
        while i >= 0 and j >= 0:  
            if board[i][j] == 'Q':  
                return False  
            i -= 1  
            j -= 1  
        # Check upper right diagonal  
        i, j = row - 1, col + 1  
        while i >= 0 and j < N:  
            if board[i][j] == 'Q':  
                return False  
            i -= 1  
            j += 1  
        return True  
  
    def backtrack(row):  
        if row == N:
```

```

    printBoard(board)

    return True

for col in range(N):

    if isSafe(row, col):

        board[row][col] = 'Q'

        if backtrack(row + 1):

            return True

        board[row][col] = '.'

    return False

```

```
backtrack(0)
```

## Program Output:

The screenshot shows a Python IDE with two windows. The main window displays the code for the N-Queens problem, and a smaller window titled 'IDLE Shell 3.13.5' shows the program's output.

**Code in the main window:**

```

exp6.1.py - /home/cyberkalai/Documents/DAA python files/exp6.1.py (3.13.5)
File Edit Format Run Options Window Help
import matplotlib.pyplot as plt

def solve_n_queens(n):
    def is_safe(board, row, col):
        for i in range(row):
            if board[i] == col or \
                board[i] - i == col - row or \
                board[i] + i == col + row:
                    return False
            return True

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

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

def visualize_solution(board, n, index):
    fig, ax = plt.subplots()
    ax.set_xticks(range(n))
    ax.set_yticks(range(n))
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_xlim(-0.5, n - 0.5)
    ax.set_ylim(-0.5, n - 0.5)
    ax.set_title(f'N = {n}, Solution #{index + 1}')

    # Draw grid
    for x in range(n):
        for y in range(n):
            color = 'white' if (x + y) % 2 == 0 else 'gray'
            ax.add_patch(plt.Rectangle((y - 0.5, n - x - 1 - 0.5), 1, 1, color))

    # Place queens
    for row, col in enumerate(board):
        ax.plot(col, n - row - 1, 'ro', markersize=20)

    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()

# Run and visualize for N = 4, 5, 8
for N in [4, 5, 8]:
    solutions = solve_n_queens(N)
    print(f'N = {N}, Total Solutions: {len(solutions)}')
    for i, sol in enumerate(solutions[:3]): # Show up to 3 solutions
        print(f'Solution {i + 1}: {sol}')
        visualize_solution(sol, N, i)

```

**Output in the IDLE Shell:**

```

Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: /home/cyberkalai/Documents/DAA python files/exp6.1.py =====

N = 4, Total Solutions: 2
Solution 1: [1, 3, 0, 2]
Solution 2: [2, 0, 3, 1]

N = 5, Total Solutions: 10
Solution 1: [0, 2, 4, 1, 3]
Solution 2: [0, 3, 1, 4, 2]
Solution 3: [1, 3, 0, 2, 4]

N = 8, Total Solutions: 92
Solution 1: [0, 4, 7, 5, 2, 6, 1, 3]
Solution 2: [0, 5, 7, 2, 6, 3, 1, 4]
Solution 3: [0, 6, 3, 5, 7, 1, 4, 2]
>>>

```

## Result:

Thus, the Nqueens program is implemented and got output executed successfully.

