

**TASK:8****Implementation of N-queen problem using backtracking algorithm.**

Implementation of N-queen problem using backtracking algorithm using python. In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

**Tools: Python**

**PROBLEM STATEMENT:****CO3 S3**

In a grand royal palace, the Queen is presented with the famous N-Queens challenge. The task is to place N queens on an  $N \times N$  chessboard so that no two queens threaten each other. This means that no two queens can share the same row, column, or diagonal.

To accomplish this, the Queen decides to follow a systematic strategy. She will place the queens one by one, checking carefully for conflicts. If a queen cannot be placed safely in a certain position, she will backtrack, moving the earlier queens to different spots, and then continue the process. By patiently repeating this method, the Queen aims to discover at least one valid arrangement that demonstrates the elegance of logic and reasoning to her court.

# **IMPLEMENTATION OF N-QUEEN PROBLEM USING BACKTRACKING ALGORITHM**

## **AIM**

To Implement N-Queen's problem by using backtracking algorithm using python.

## **ALGORITHM**

1. Initialize board: Create an  $N \times N$  chessboard and set all cells to empty.
2. Start at first column: Begin placing queens in the first column.
3. Place queen: Attempt to place a queen in the current column, starting from the first row.
4. Check safety: Before placing, check if the cell is safe (no other queen in the same row, column, or diagonal).
5. Place if safe: If the position is safe, place the queen in that cell.
6. Move to next column: Recursively attempt to place a queen in the next column.
7. Backtrack if needed: If no safe position is found in a column, remove the previously placed queen (backtrack) and try the next row in the previous column.
8. Repeat steps 3–7: Continue until all queens are placed successfully or all possibilities are exhausted.
9. Solution found: Once  $N$  queens are placed safely, stop recursion and record this arrangement as a solution.
10. Display solution: Print the board showing the positions of the  $N$  queens.

## PROGRAM

### N-Queens Problem

```
def is_safe(board, row, col, N):  
    # Check this row on the left  
    for i in range(col):  
        if board[row][i] == 1:  
            return False  
    # Check upper diagonal on left side  
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):  
        if board[i][j] == 1:  
            return False  
    # Check lower diagonal on left side  
    for i, j in zip(range(row, N, 1), range(col, -1, -1)):  
        if board[i][j] == 1:  
            return False  
    return True  
  
def solve_nqueens_one_solution(board, col, N):  
    if col >= N:  
        return True # Found one solution  
    for row in range(N):  
        if is_safe(board, row, col, N):  
            board[row][col] = 1  
            if solve_nqueens_one_solution(board, col + 1, N):  
                return True  
            board[row][col] = 0 # backtrack  
    return False  
  
def print_solution(board, N):  
    for row in board:  
        print(" ".join('Q' if x else '.' for x in row))
```

```
# Example: Solve 4-Queens
```

```
N = 4
```

```
board = [[0]*N for _ in range(N)]
```

```
if solve_nqueens_one_solution(board, 0, N):
```

```
    print(f"One solution for {N}-Queens:")
```

```
    print_solution(board, N)
```

```
else:
```

```
    print(f"No solution exists for {N}-Queens")
```

## OUTPUT

One solution for 4-Queens:

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

## RESULT

Thus, the Implementation of N-queen problem using backtracking algorithm using Python was successfully executed and output was verified.