Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	S. Y. B.Tech (Computer Engineering)
Course:	Analysis of Algorithm Laboratory
Course Code:	DJ19CEL404
Experiment No.:	08

AIM: **TO IMPLEMENT N QUEEN'S PROBLEM**

THEORY:

N_QUEEN

- The N Queens problem is a classic problem in computer science and mathematics.
- The problem asks for the number of ways N queens can be placed on an N×N
- chessboard such that no two queens threaten each other.
  - In other words, the queens cannot share the same row, column, or diagonal. One approach to solving this problem is to use backtracking. The algorithm places one queen in each column, starting from the leftmost column.
  - Once a queen is placed, the algorithm checks if it is under attack by any of the previously placed queens. If it is not under attack, the algorithm moves to the next column and places another queen.
  - If a queen cannot be placed in a column without being under attack, the algorithm backtracks to the previous column and tries a different row for that column.
- **Algorithm:**

```
1. Initialize an empty chessboard of size NxN.
2. Start with the leftmost column and place a queen in the first row of that
column.
3. Move to the next column and place a queen in the first row of that column.
4. Repeat step 3 until either all N queens have been placed or it is
impossible to place a
queen in the current column without violating the rules of the problem.
5. If all N queens have been placed, print the solution.
6. If it is not possible to place a queen in the current column without
violating the rules of
the problem, backtrack to the previous column.
7. Remove the queen from the previous column and move it down one row.
8. Repeat steps 4-7 until all possible configurations have been tried.
```

CODE:

```
#define N 8
#include <stdbool.h>
#include <stdio.h>
void printSolution(int board[N][N])
{
```

```
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;
    return true;
}
bool solveNQUtil(int board[N][N], int col)
{
    if (col >= N)
        return true;
    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col))
        {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}
bool solveNQ()
{
    int board[N][N] = {{0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0}};
    if (solveNQUtil(board, 0) == false)
```

```
    {
        printf("Solution does not exist");
        return false;
    }
    printSolution(board);
    return true;
}
int main()
{
    solveNQ();
    return 0;
}
```

OUTPUT:

```
exe' '--interpreter=mi'
 1  0  0  0  0  0  0  0
 0  0  0  0  0  0  1  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  1
 0  1  0  0  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  0  0  1  0  0
 0  0  1  0  0  0  0  0
PS C:\Users\Jadhav\Desktop\BTech\4th sem\AOA\Prac\Code>
```

CONCLUSION:

- Thus, we implemented the code to solve N Queens Problem. Here, 8 Queens Problem.