Assignment No: 11

Aim:

A classic problem that can be solved by backtracking is called the eight queens problem, which comes from game of chess. Chess board consists of 64 square arranged in 8 by 8 grid, but this is not releavant for present problem. The queen can move as far as she wants in any direction as long as she follows a straight line. Vertically, horizontally or diagonally. Write C++ program for generating all possible configurations for 4-queen's problem.

Theory:

This problem is to find an arrangement of N queens on chess board, such that no queen can attack any other queen on board. The class queen can attack in any direction as horizontal vertical, diagonal ways. A binary matrix is used to display the positions of N-Queens where no-queen can attack other queens.

Algorithm:

is valid (board, row, col)

Input:

The chess board, row & column of board.

Output:

True when placing a queen in row & place position is valid or not.

Solve NQueen (board, col)

Input:

Chess board, column where queen is placed

Output:

Position matrix where queen is placed.

Program Code :-

```
#include <iostream>
using namespace std;
#define N 8

void printBoard(int
board[N][N])
{
    for (int i = 0; i < N;
i++)
    {
        for (int j = 0; j < N;
j++)
            cout << board[i][j]
<< " ";
        cout << endl;
    }
}

bool isValid(int
board[N][N], int row, int
col)
{
    for (int i = 0; i < col;
i++)
        if (board[row][i])
            return false;
```

```
    for (int i = row, j = col;
i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    for (int i = row, j = col;
j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;
    return true;
}

bool solveNQueen(int
board[N][N], int col)
{
    if (col >= N)
        return true;
    for (int i = 0; i < N;
i++)
    {
        if (isValid(board, i,
col))
        {
            board[i][col] = 1;
            if
(solveNQueen(board, col
+ 1))
                return true;

            board[i][col] = 0;
        }
```

```
    }
    return false;
}

bool checkSolution()
{
    int board[N][N];
    for (int i = 0; i < N;
i++)
        for (int j = 0; j < N;
j++)
            board[i][j] = 0;

    if (solveNQueen(board,
0) == false)
    {
        cout << "Solution
does not exist";
        return false;
    }
    printBoard(board);
    return true;
}

int main()
{
    checkSolution();
}
```

Program Output :-

```
"C:\Users\prath\OneDrive\Desktop\DSAsahil\SCOA68_Sahil Thete_DSA_Assignment_11.exe"
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

Process returned 0 (0x0)   execution time : 0.969 s
Press any key to continue.
```