**Name**: Ravi Jamanbhai Makwana

**Roll No**: 21BCP418

Div-6, G-12

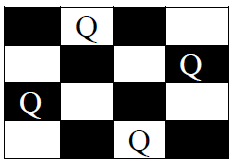
**Assignment-8**

**Aim:**

Write a program to solve the 𝒏 queens’ problem using backtracking. Here, the task is to place 𝒏 chess queens on an 𝑛 x 𝑛 board so that no two queens attack each other.

**Theory:**

* The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, the following is a solution for the 4 Queen problem:



* The expected output is in form of a matrix that has ‘1’s for the blocks where queens are placed and the empty spaces are represented by ‘0’s.

**Algorithm:**

* Initialize an empty chessboard of size NxN.
* Start with the leftmost column and place a queen in the first row of that column.
* Move to the next column and place a queen in the first row of that column.
* 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.
* If all N queens have been placed, print the solution.
* 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.
* Remove the queen from the previous column and move it down one row.
* Repeat steps 4-7 until all possible configurations have been tried.

**Program:**

#include <stdio.h>

#include <stdbool.h>

#define n 4

int ChessBoard[n][n];

//function to print the Chess Board

//(prints 0 at empty places, prints 1 at places where Queens are present)

void printChessBoard(int ChessBoard[n][n]){

    for (int i=0;i<n;i++){

        for (int j=0;j<n;j++){

            printf("%d ",ChessBoard[i][j]);

        }

        printf("\n");

    }

}

//function to check if the Queen that we are going to place at [row,col] (not yet placed) is safe to place or not

//since we are placing the Queens column by column, starting from the 0th column, so

//before we place a new Queen on the chess board, we have to check at 3 line of sight of the current Queen position to check for any attacking Queens that might be already placed:

//          - at the Left side of the Queen

//          - Top Left diagonal

//          - Bottom Left diagonal

//This function is called when there are already 'col' Queens placed in columns 0 to (col-1)

bool isSafe(int ChessBoard[n][n], int row, int col){

    int i,j;

    //check on Left side

    for (i=0;i<col;i++){

        if (ChessBoard[row][i]){

            return false;   //if any Queen is present on the left side

        }

    }

    //check on Top Left diagonal

    for (i=row,j=col ; (i>=0 && j>=0) ; i--,j--){

        if (ChessBoard[i][j]){

            return false;

        }

    }

    //check on Bottom Left diagonal

    for (i=row,j=col ; i<n && j>=0 ; i++,j--){

        if (ChessBoard[i][j]){

            return false;

        }

    }

    //if no attacking Queen is found in all 3 cases

    return true;

}

//function that is called recursively to solve the NQueens problem, using Backtracking

bool solve(int ChessBoard[n][n], int col){

    //base case (if all thet Queens are placed)

    if (col >= n){

        return true;

    }

    //else consider the given col and try placing the queen one by one in each row of that column

    for (int i=0;i<n;i++){

        //now check if the queen can be placed at this position or not. i.e. at (i,col)

        if (isSafe(ChessBoard, i, col)){

            //place the Queen at that spot

            ChessBoard[i][col] = 1;

            //use recursion for placing other Queens

            if (solve(ChessBoard, col + 1)){

                return true;

            }

            //if placing the Queen at that spot (i,col) doesn't lead to a solution,

            // or simply if we can't place the Queen there,

            // then, remove the Queen from position (i,col)

            // BACKTRACK

            ChessBoard[i][col] = 0;

        }

    }

    //if Queen cannot be placed in any row of the column col, then return false

    return false;

}

int main(){

    int ChessBoard[n][n];

    //making chess board(initialize with all 0s)

    for (int i=0;i<n;i++){

        for (int j=0;j<n;j++){

            ChessBoard[i][j] = 0;

        }

    }

    if (solve(ChessBoard, 0)){

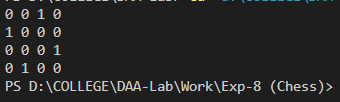
        printChessBoard(ChessBoard);

    }

    else printf("No solution exists for n = %d", n);

}

**Output:**



**Analysis:**

**Time Complexity Analysis:**

**Time complexity**: **O(N!)**: The first queen has N placements, the second queen must not be in the same column as the first as well as at an oblique angle, so the second queen has N-1 possibilities, and so on, with a time complexity of O(N!).

**Auxiliary Space: O(N2)**