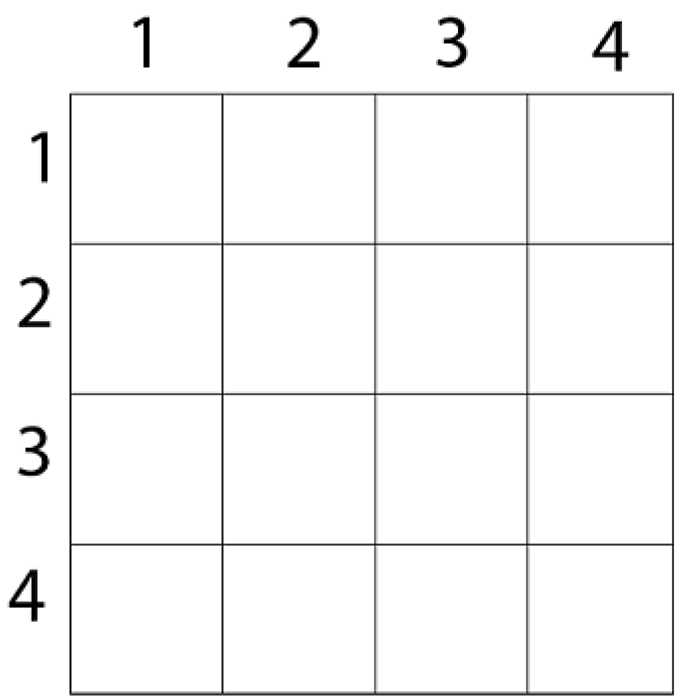
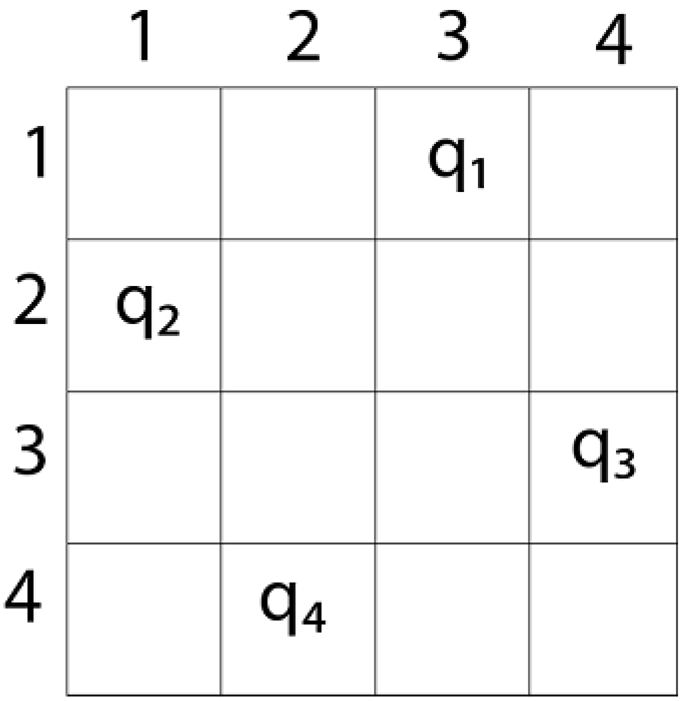
Name	Anand Tiwari
UID no.	2021700068
Experiment No.	7

AIM:	Solving N-Queens problem using backtracking
Program 1	
PROBLEM STATEMENT:	To implement N-Queens using backtracking
	N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal. It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem. Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.  4x4 chessboard

Since, we have to place 4 queens such as q_1 q_2 q_3 and q_4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.



It can be seen that all the solutions to the 4 queens problem can be represented as 4 - tuples (x_1, x_2, x_3, x_4) where x_i represents the column on which queen "q_i" is placed.

One possible solution for 8 queens problem is shown in fig:

	1	2	3	4	5	6	7	8
1				q_1				
2						q_2		
3								q_3
4		q_4						
5							q_5	
6	q_6							
7			q_7					
8					q_8			

Place (k, i) returns a Boolean value that is true if the kth queen can be placed in column i. It tests both whether i is distinct from all previous costs $x_1, x_2, \ldots x_{k-1}$ and whether there is no other queen on the same diagonal.

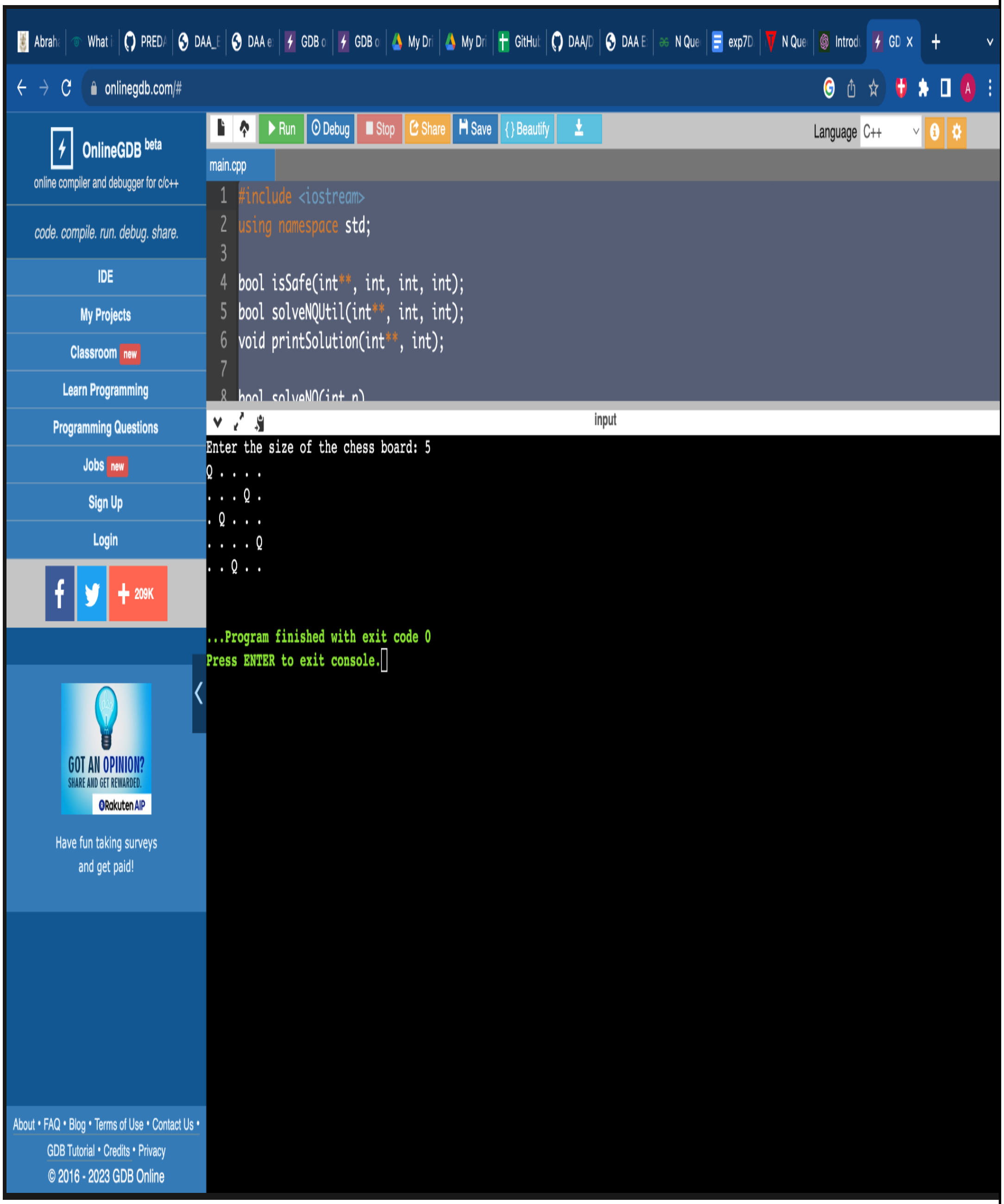
Using place, we give a precise solution to then n- queens problem.

PROGRAM:

```
#include <iostream>
using namespace std;

bool isSafe(int**, int, int, int);
bool solveNQUtil(int**, int, int);
void printSolution(int**, int);

bool solveNQ(int n)
{
    int** board = new int*[n];
    for (int i = 0; i < n; ++i) {
        board[i] = new int[n];
        for (int j = 0; j < n; ++j) {
            board[i][j] = 0;
        }
    }

    if (!solveNQUtil(board, n, 0)) {
        cout << "Solution does not exist" << endl;
        return false;
    }

    printSolution(board, n);

    for (int i = 0; i < n; ++i) {
        delete[] board[i];
    }
    delete[] board;

    return true;
}

bool isSafe(int** board, int n, int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++) {
        if (board[row][i]) {
            return false;
        }
    }

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
    }
```

```
    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < n; i++, j--) {
        if (board[i][j]) {
            return false;
        }
    }

    return true;
}

bool solveNQUtil(int** board, int n, int col)
{
    if (col >= n) {
        return true;
    }

    for (int i = 0; i < n; i++) {
        if (isSafe(board, n, i, col)) {
            board[i][col] = 1;

            if (solveNQUtil(board, n, col + 1)) {
                return true;
            }

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

    return false;
}

void printSolution(int** board, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (board[i][j]) {
                cout << "Q ";
            } else {
                cout << ". ";
            }
        }
        cout << endl;
    }
}

int main()
{
    int n;
    cout << "Enter the size of the chess board: ";
    cin >> n;
```

	`solveNQ(n);`
	`return 0;`
	`}`

OUTPUT	

RESULT: Successfully understood NQueens algorithm and implemented it in C program.