NAME:	Shubham Solanki
UID:	2022301015
SUBJECT	Design and Analysis of Algorithms
EXPERIMENT NO:	07
AIM:	To implement Backtracking (N Queen Problem)

Algorithm:	**Backtracking Algorithm**
	
	function solveNQueens(board, col, n):
	  if col >= n:
	  print board
	  return true
	  for row from 0 to n-1:
	  if isSafe(board, row, col, n):
	  board[row][col] = 1
	  if solveNQueens(board, col+1, n):
	  return true
	  board[row][col] = 0
	  return false
	
	function isSafe(board, row, col, n):
	  for i from 0 to col-1:
	  if board[row][i] == 1:
	  return false
	  for i,j from row-1, col-1 to 0, 0 by -1:
	  if board[i][j] == 1:
	  return false
	  for i,j from row+1, col-1 to n-1, 0 by 1, -1:
	  if board[i][j] == 1:
	  return false
	  return true
	
	board = empty NxN chessboard

	solveNQueens(board, 0, N)
Code:	

```
#include <stdio.h>
#include <stdbool.h>

void printSolution(int n, int board[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%c ", board[i][j] ? 'Q' : '.');
        }
        printf("\n");
    }
    printf("\n");
}

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

    // Check the left side of the row
    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;
}

void solveNQueensUtil(int n, int board[n][n], int
col) {
    if (col == n) {
        printSolution(n, board);
        return;
    }

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

void solveNQueens(int n) {
    int board[n][n];

    // Initialize the board to all 0s
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
            board[i][j] = 0;
        }
    }
    solveNQueensUtil(n, board, 0);
}

int main() {
    int n = 0;
    printf("\nEnter the dimension of the chessboard
: ");
    scanf("%d",&n);
    solveNQueens(n);
    return 0;
}
```

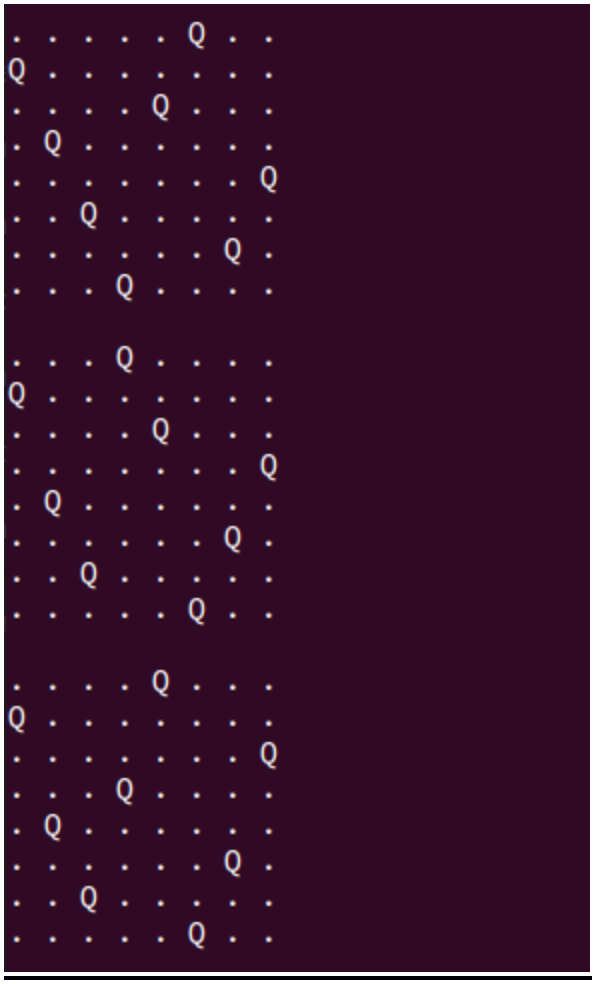
Output:

```
shubham@shubham-virtual-machine:~/semester4/daa/experiments/exp 7$ gcc nQueenProblem.c
shubham@shubham-virtual-machine:~/semester4/daa/experiments/exp 7$ ./a.out

Enter the dimension of the chessboard : 8
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .

Q . . . . . . .
. . . . . . Q .
. . . Q . . . .
. . . . . Q . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
. . Q . . . . .

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

	 . . . . . Q . . Q . . . . Q . . . . Q . . . . Q . . Q . . . . Q . . . . Q . . . . Q . . . . Q . . . . Q . . . . Q . Q . . . . Q . . . Q . . . . Q . . . . . . Q . . . Q . . . . Q . . . Q . . . . . Q . . . . Q . . . Q . . . . Q . .
Conclusion:	Thus by the end of this experiment we have learnt about Backtracking. Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time. A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output. We also have also implemented the N Queen Problem