Name: Rishi Tiku

Class: SE DS

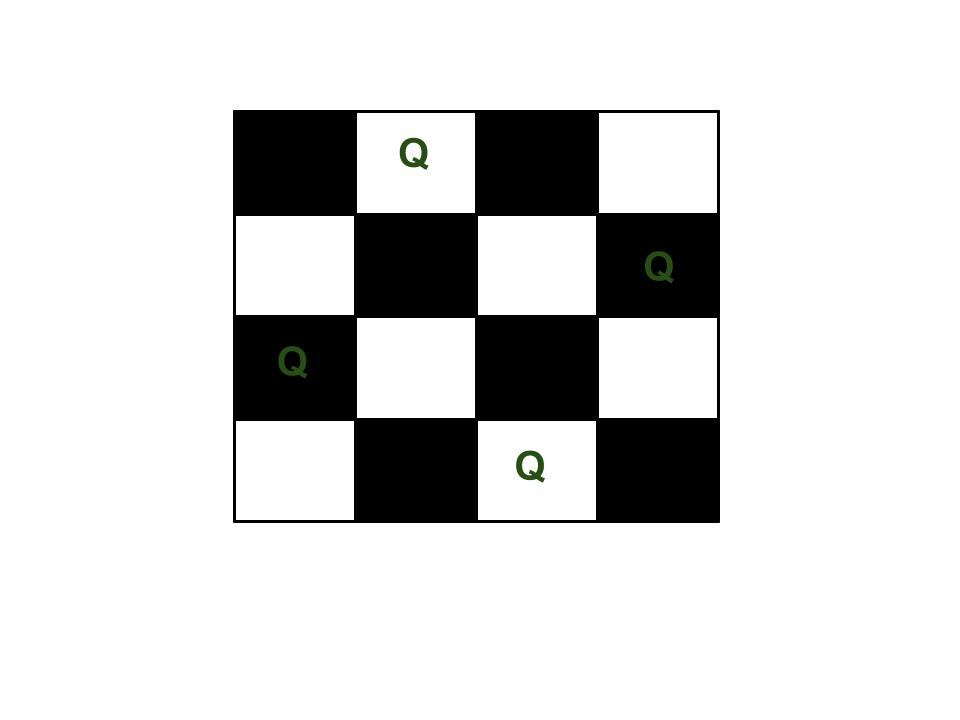
UID: 2021700067

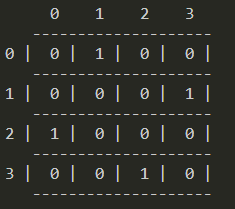
Subject: DAA LAB

Experiment: 8

**Aim:** To implement backtracking to solve N-Queens problem

**Problem:** 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 1s for the blocks where queens are placed and the empty spaces are represented by 0s.

For example: 

**Algorithm for N queen problem:-**

* 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.

**Pseudo-code implementation:**

*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 <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

*void* Display(*int* *N*, *int* *A*[*N*][*N*])

{

    printf("\n     ");

    for(*int* i = 0; i<*N*; i++)

    {

        printf("%d    ", i);

    }

    printf("\n   ");

    for(*int* i = 0; i<*N*; i++)

    {

        printf("-----");

    }

    printf("\n");

    for (*int* i = 0; i < *N*; i++)

    {

        printf("%d | ", i);

        for (*int* j = 0; j < *N*; j++)

        {

            printf(" %d | ", *A*[i][j]);

        }

        printf("\n   ");

       for(*int* k = 0; k<*N*; k++)

        {

            printf("-----");

        }

        printf("\n");

    }

}

bool isSafe(*int* *N*, *int* *board*[*N*][*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;

}

/\* A recursive function to solve N

Queen problem using DFS\*/

bool DFS(*int* *N*, *int* *board*[*N*][*N*], *int* *depth*)

{

    /\* base case: If all queens are placed

    then print \*/

    if (*depth* >= *N*)

    {

*char* ch;

        //printSolution(N, board);

        printf("\nSolution: \n");

        Display(*N*, *board*);

        printf("\nPress Enter Key for Next solution.");

        fflush(stdin);

        scanf("%c", &ch);

        printf("\n");

        return false;

    }

    /\* Consider this column and try placing

    this queen in all rows one by one \*/

    for (*int* i = 0; i < *N*; i++) {

        /\* Check if the queen can be placed on

        board[i][depth] \*/

        if (isSafe(*N*, *board*, i, *depth*)) {

            {/\* Place this queen in board[i][depth] \*/

*char* ch;

*board*[i][*depth*] = 1;

            }

            DFS(*N*, *board*, *depth* + 1);

            /\* If placing queen in board[i][col]

            doesn't lead to a solution, then

            remove queen from board[i][depth] \*/

*board*[i][*depth*] = 0; // BACKTRACK

        }

    }

    /\* If the queen cannot be placed in any row in

        this column col then return false \*/

    return false;

}

bool solveNQ(*int* *N*)

{

*int* board[*N*][*N*];

    for(*int* i = 0 ; i<*N*; i++)

    {

        for(*int* j = 0; j<*N*; j++)

        {

            board[i][j] = 0;

        }

    }

    if (*N* == 2 || *N* == 3) {

        printf("Solution does not exist");

        return false;

    }

    DFS(*N*,board, 0);

    return true;

}

// main function

*int* main()

{

*int* N = 0;

    printf("Input N: \n");

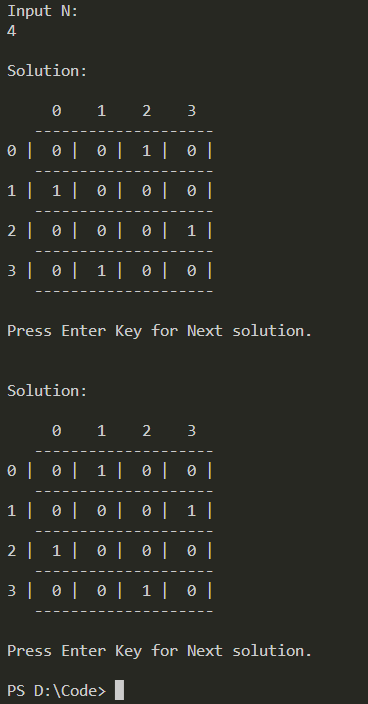
    scanf("%d", &N);

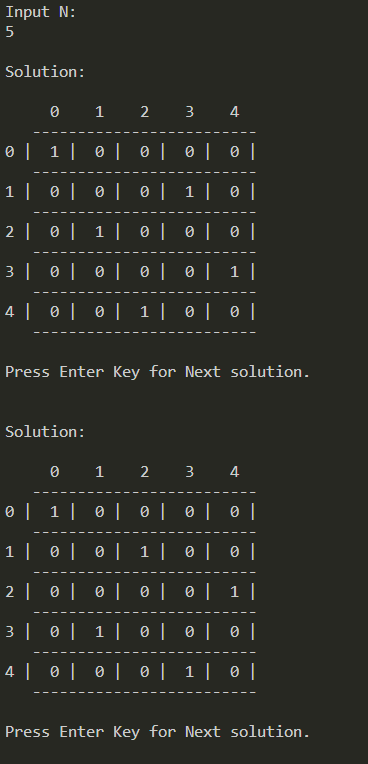
    solveNQ(N);

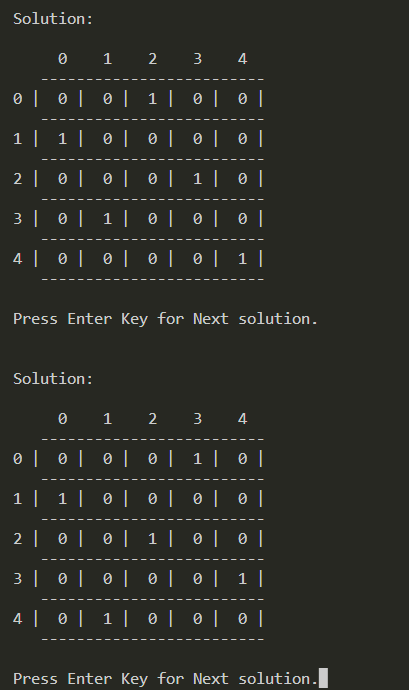
    return 0;

}

**Result:**

****

****

**…10 solutions in total for N = 5**

**Time Complexity:**O(N^2 \* N!)   
**Auxiliary Space:**O(N)

**Conclusion**

The N-queens problem is solved by implementing backtracking technique.