EXPERIMENT 8

20CP209P – Design and Analysis of Algorithm Lab

Aim:

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. For example, following is a solution for the 4 Queen' problem

Code:

N Queens Problem:

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

#define fr(i, a, b) for (int i = a; i < b; i++)
#define N 4
// #define N 5
// #define N 6
// #define N 7

int total_sol_count = 0;

void print_sol(int board[N][N]);
bool is_safe(int board[N][N], int row, int col);
bool solveNQUtil(int board[N][N], int col);

int main(void)
{
   // int board[N][N] = {
   //     {0, 0, 0, 0},
   //     {0, 0, 0, 0},
   //     {0, 0, 0, 0},
   //     {0, 0, 0, 0}
   // };
   int board[N][N];
   fr (i, 0, N)
   {
      fr(j, 0, N) board[i][j] = 0;
   }

   if (solveNQUtil(board, 0) == false)
   {
      printf("Solution does not exist");
      return 0;
   }

   // print_sol(board);
```

```
    printf("Total solutions found: %d\n", total_sol_count);

    return 0;
}

void print_sol(int board[N][N])
{
    fr(i, 0, N)
    {
        fr (j, 0, N)
        {
            printf("%d", board[i][j]);
        }
        printf("\n");
    }
}

bool is_safe(int board[N][N], int row, int col)
{
    int i, j;
    fr (i, 0, col)
    {
        if (board[row][i])
        {
            return false;
        }
    }

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
    {
        if (board[i][j])
        {
            return false;
        }
    }

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

    return true;
}

bool solveNQUtil(int board[N][N], int col)
{
```

```
    if (col >= N)
    {
        total_sol_count++;

        printf("\nsol\n");
        print_sol(board);
        return true;
    }
    // changes might be required here

    // Flag to track if any solution is found from this column onwards.
    // Initialize to false. It will be set to true if any recursive call
    // down the line finds a solution.
    bool res = false;

    for (int i = 0; i < N; i++)
    {
    // if it is safe to place the queen at position i, col -> place it
        if (is_safe(board, i, col))
        {

            board[i][col] = 1;
            // printf("row: %d\n", i);
            // print_sol(board);
            // printf("\n");
            // if (solveNQUtil(board, col + 1))
            // {
            //     return true;
            // }

            // Recur to place the rest of the queens for the next column (col + 1).
            // Crucially, we use 'res = solveNQUtil(...) || res;'
            // This calls the function for the next column AND combines its result
            // (true if a solution was found down that path) with any previous
            // results found by trying other rows in this *current* column.
            // We do NOT return immediately.
            res = solveNQUtil(board, col + 1) || res;

            // backtrack if the above condition is false
            board[i][col] = 0; // BACKTRACK
        }
    }

    // return false;

    // Return the final result 'res'. It will be true if any placement
    // in this column 'col' led to at least one solution down the recursion path.
    // It will be false if no placement in this column led to any solution.
    return res;
}
```

Analysis:

N- Queens.

```
#define N 4
total sol count = 0
board [N][N] = {{0, 0, 0, 0}}

for col = 0 to N-1
    for row = 0 to N-1
        if is safe (board, row, col)
            set board [row] [col] = 1
            if SolveNQutil(board, col+1)
                == true
                return true
            board [row][col] = 0
print_sol(board).
```

Worst case

$O(N^2)$	
$O(N^2)$	set N x N Matrix
$O(N)$	check and try all rows
$O(N!)$	
$O(1)$	explore all queen place
$O(N^2)$	done on rejection. Print N x N

Total $O(N \times N!)$

TC = $O(N \times N!)$ Full tree explored

Average case = $O(N!)$

Best case = $O(N^2)$

Scanned with CamScanner

Output:

```
PS B:\sem4\23bcp153_daa\lab8> gcc nqueens.c -o nqueens
PS B:\sem4\23bcp153_daa\lab8> ./nqueens

sol
0010
1000
0001
0100

sol
0100
0001
1000
0010
Total solutions found: 2
PS B:\sem4\23bcp153_daa\lab8>
```