

Experiment 8

Aim: To implement n-queens problem.

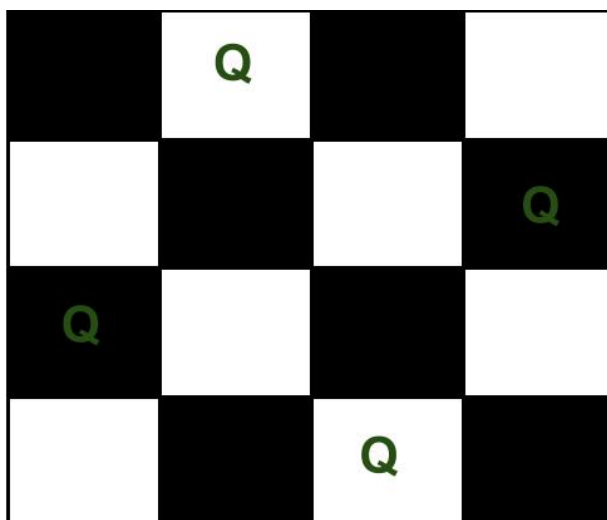
Theory: The n queens problem is a classic problem in computer science and mathematics that involves placing n chess queens on an $n \times n$ chessboard such that no two queens threaten each other. In other words, no two queens can be placed on the same row, column, or diagonal.

The problem has a simple solution for small values of n , but it becomes increasingly challenging as n grows larger. The problem can be solved using various algorithms, including brute force search, backtracking, and heuristic methods. The solution to the problem has applications in various fields, including computer science, mathematics, and operations research.

The n queens problem is a popular topic for research and study in computer science and mathematics and has many variations and extensions that continue to inspire new research and discoveries.

Example:

following is a solution for 4 Queen problem.



Algorithm:

$\xrightarrow{\text{row}}$ $\xrightarrow{\text{column}}$
 Algorithm Place(k, i)
 { // returns true if Q can be placed in k^{th} row
 // $x[]$ is a global array whose $(k-1)$ values have been placed
 for $j = 1$ to $k-1$
 if ($x[j] == i$ or ($\text{Abs}(x[j] - i) == \text{Abs}(j - k)$))
 return False
 return True }

Algorithm NQueen(k, n)
 { // using backtracking this algo prints all possible placements of n queens
 for $i = 1$ to n
 if Place(k, i)
 $x[k] = i$
 if $k == n$
 Print(x)
 else
 NQueen($k+1, n$) }

Code:

```

#define N 4
#include <stdbool.h>
#include <stdio.h>
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
        printf("\n");
    }
}

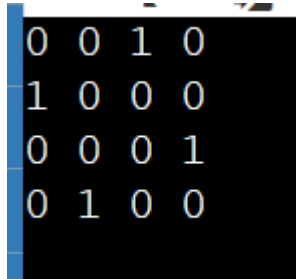
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;
    for (i = 0; i < col; i++)

```

```
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)
return true;
for (int i = 0; i < N; i++) {
if (isSafe(board, i, col)) {
board[i][col] = 1;
if (solveNQUtil(board, col + 1))
return true;
board[i][col] = 0;
}
}
return false;
}
bool solveNQ()
{
int board[N][N] = { { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 },
                    { 0, 0, 0, 0 } };
if (solveNQUtil(board, 0) == false) {
printf("Solution does not exist");
return false;
}
printSolution(board);
return true;
}
```

```
int main()
{
    solveNQ();
    return 0;
}
```

Output:



```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

Conclusion: Hence we have successfully implemented n queens problem.