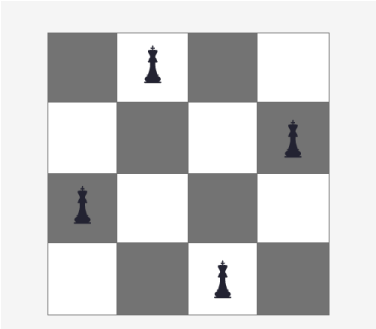
AOA EXPERIMENT 11

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: To study and implement the N Queen Problem using a backtracking approach.

Theory:
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 the form of a matrix that has ‘Q‘s for the blocks where queens are placed and the empty spaces are represented by ‘.’ .
For example, the following is the output matrix for the above 4-Queen solution.

. Q . .

. . . Q

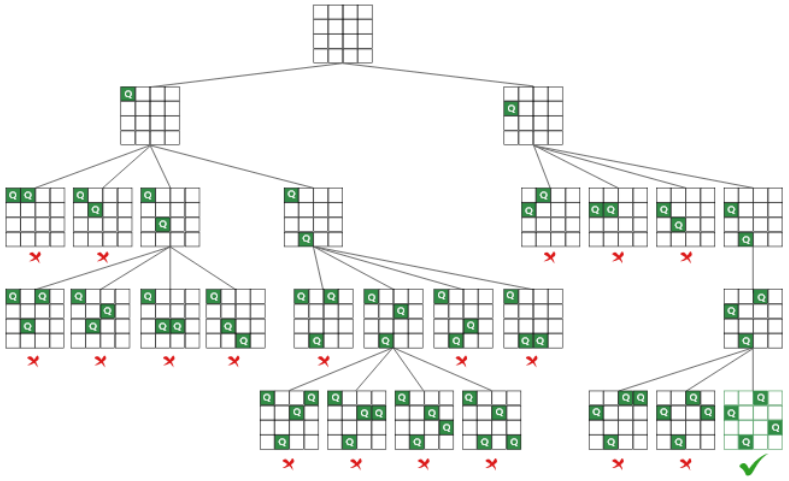
Q . . .

. . Q .

N Queen Problem using Backtracking:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with

already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.
Below is the recursive tree of the above approach:



Recursive tree for N Queen problem
Follow the steps mentioned below to implement the idea:

- Start in the leftmost column
- If all queens are placed return true
- Try all rows in the current column. Do the following for every row.
    - If the queen can be placed safely in this row
        - Then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
        - If placing the queen in [row, column] leads to a solution then return true.
        - If placing queen doesn't lead to a solution then unmark this [row, column] then backtrack and try other rows.
    - If all rows have been tried and a valid solution is not found, return false to trigger backtracking.

Program:

```
#define N 4
#include <stdbool.h>
#include <stdio.h>

// A utility function to print solution
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
            if(board[i][j])
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
}

// A utility function to check if a queen can
// be placed on board[row][col]. Note that this
// function is called when "col" queens are
// already placed in columns from 0 to col -1.
// So we need to check only left side for
// attacking queens
bool isSafe(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 utility function to solve N
// Queen problem
bool solveNQUtil(int board[N][N], int col)
{
    // Base case: If all queens are placed
    // then return true
    if (col >= N)
        return true;

    // 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][col]
        if (isSafe(board, i, col)) {

            // Place this queen in board[i][col]
            board[i][col] = 1;

            // Recur to place rest of the queens
            if (solveNQUtil(board, col + 1))
                return true;

            // If placing queen in board[i][col]
            // doesn't lead to a solution, then
            // remove queen from board[i][col]
            board[i][col] = 0; // BACKTRACK
        }
    }

    // If the queen cannot be placed in any row in
    // this column col  then return false
    return false;
}

// This function solves the N Queen problem using
```

```
// Backtracking. It mainly uses solveNQUtil() to
// solve the problem. It returns false if queens
// cannot be placed, otherwise, return true and
// prints placement of queens in the form of 1s.
// Please note that there may be more than one
// solutions, this function prints one  of the
// feasible solutions.
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;
}

// Driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

Output:

```
PS C:\Users\arhaa\OneDrive\Desktop\AOA> cd "c:\Users\arhaa\OneDr
 .\nqueen }
. . Q .
Q . . .
. . . Q
. Q . .
PS C:\Users\arhaa\OneDrive\Desktop\AOA>
```

Conclusion: Thus we have successfully implemented The N Queen Problem using Backtracking approach.