Government College of Engineering, Jalgaon **(An Autonomous Institute of Government of Maharashtra)**	
Name :	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25
Subject : CO356U DAA Lab	**Course Teacher** : Mr. Vinit Kakde
Date of Performance :	**Date of Completion :**

Practical no.12

AIM: To implement the N-Queen problem using backtracking in C.

THEORY:

The N-Queen Problem is a classic combinatorial problem in which the goal is to place N queens on an N x N chessboard such that no two queens threaten each other. This means:

- No two queens can be in the same row,
- No two queens can be in the same column,
- No two queens can be on the same diagonal.

The problem is typically solved using backtracking, which tries to build the solution incrementally and abandons (backtracks) as soon as it determines the current path won't lead to a solution.

ALGORITHM:

1. Start in the leftmost column (column 0).
2. Try placing a queen in each row of the current column, and check if it's **safe**.
3. If it is safe, mark the position and recursively try placing the next queen in the next column.
4. If placing a queen leads to a dead end, **backtrack** by removing the queen and trying the next position.
5. Repeat until all N queens are placed.

To check if a queen can be placed at board[row][col]:

- Check the same column in previous rows.
- Check the upper-left diagonal.
- Check the upper-right diagonal.

Complexity:

- **Time Complexity:** O(N!) in the worst case.
- **Space Complexity:** O(N″) for the board (can be optimized to O(N)).

Advantages:

- Good example of **backtracking**.
- Can be extended to solve variations like:
    - Placing other pieces with constraints,
    - Optimization-based problems.

Disadvantages:

- Computationally expensive for large N.
- May not be efficient without pruning or heuristics.

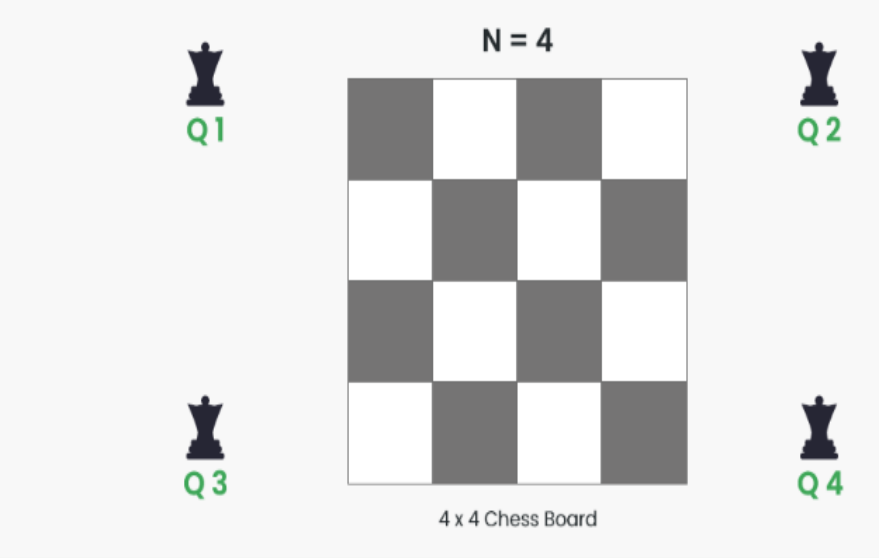
Applications:

- AI and constraint satisfaction problems.
- Optimization and puzzle-solving.
- Robotics path planning and scheduling.
- Problems involving permutations and safety constraints.

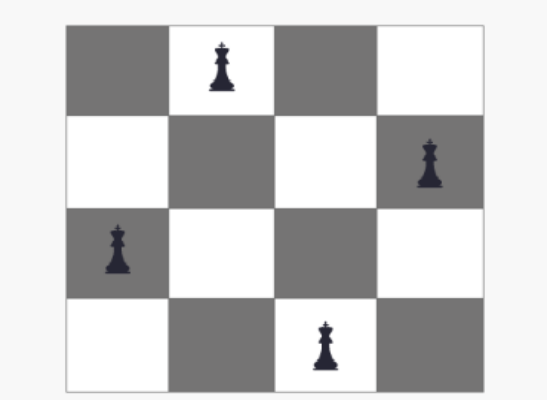
Example:

Given an integer **n**, the task is to find the solution to the **n-queens problem**, where **n** queens are placed on an **n*n** chessboard such that no two queens can attack each other.

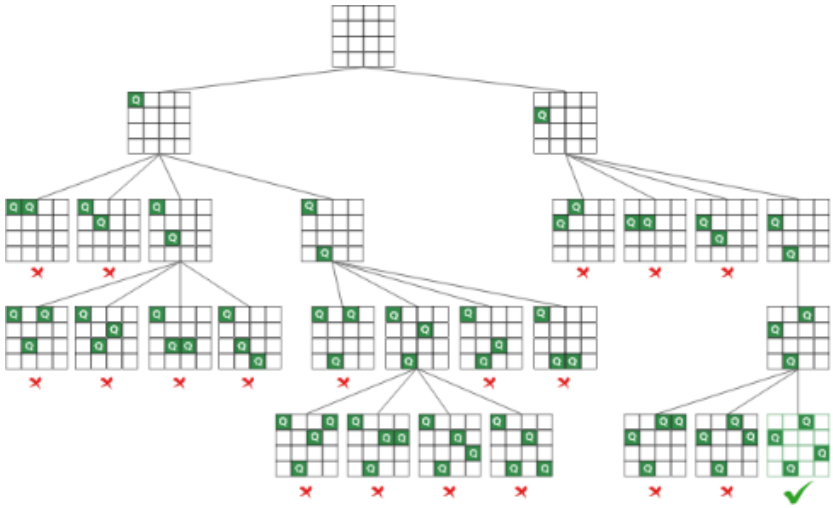
The **N** Queen is the problem of placing **N** chess queens on an **N×N** chessboard so that no two queens attack each other.



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



Recursive tree of the above approach :



PROGRAM:

```
import java.util.Scanner;

public class NQueens {
    private static final int MAX = 20;
    private static int[] board = new int[MAX];
    private static int n;
    private static int solutionCount = 0;
```

```
     private static void printSolution() {
       solutionCount++;
       System.out.println("Solution " + solutionCount + ":");
       for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
           if (board[i] == j)
               System.out.print("Q ");
           else
               System.out.print(". ");
       }
       System.out.println();
   }

   System.out.println("Queen positions (row : column):");
   for (int i = 0; i < n; i++) {
       System.out.println("Row " + (i + 1) + " -> Column " + (board[i] + 1));
   }
   System.out.println();
}


private static boolean isSafe(int row, int col) {
   for (int i = 0; i < row; i++) {
       if (board[i] == col || Math.abs(board[i] - col) == Math.abs(i - row))
           return false;
   }
   return true;
}


private static void solveNQueens(int row) {
   if (row == n) {
       printSolution();
       return;
   }

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

public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter the number of queens (N): ");
        n = scanner.nextInt();

        if (n < 1 || n > MAX) {
            System.out.println("Please enter a valid number (1 to " + MAX + ").");
            return;
        }

        System.out.println("\nAll possible solutions to the " + n + "-Queens problem:\n");
        solveNQueens(0);

        if (solutionCount == 0) {
            System.out.println("No solutions exist for N = " + n + ".");
        } else {
            System.out.println("Total solutions found: " + solutionCount);
        }
    }
}
```

OUTPUT:

```
DELL@DESKTOP-ANHHJSF MINGW64 /d/JAVA learning/JAVA
$ javac NQueens.java

DELL@DESKTOP-ANHHJSF MINGW64 /d/JAVA learning/JAVA
$ java NQueens
Enter the number of queens (N): 4

All possible solutions to the 4-Queens problem:

Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .
Queen positions (row : column):
Row 1 -> Column 2
Row 2 -> Column 4
Row 3 -> Column 1
Row 4 -> Column 3

Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .
Queen positions (row : column):
Row 1 -> Column 3
Row 2 -> Column 1
Row 3 -> Column 4
Row 4 -> Column 2

Total solutions found: 2
```

QUESTIONS:

Q1: Is the N-Queens problem solvable for all values of N?
A: No, there are no solutions for N = 2 or N = 3.

Q2: What algorithm is commonly used to solve the N-Queens problem?
A: Backtracking.

Q3: Can the N-Queens problem be solved using recursion?
A: Yes, recursive backtracking is the standard approach.

Q4: Does the N-Queens problem have more than one solution?
A: Yes, many values of N have multiple valid solutions.

Q5: How is a queen's attack path checked in the solution?
A: By checking the same column and both diagonals.

CONCLUSION:

The N-Queen Problem is a foundational problem in computer science and algorithms, especially in the area of backtracking. It teaches how to approach problems using recursion, decision trees, and state-space search, and is widely used in education and interviews to understand problem-solving strategies.

Sign of Course Teacher
Mr. Vinit Kakde