LP II Artificial Intelligence

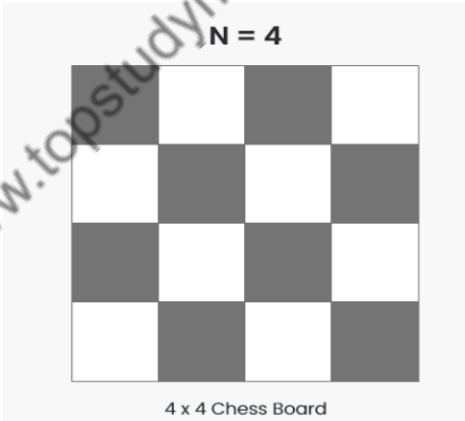
Practical No 4

Problem Statement : Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and **Backtracking for n-queens problem** or a graph coloring problem.

Solution:

We need to place **N queens** on an **N×N chessboard** such that **no two queens attack each other**. Queens can attack in:

- Same **row**
- Same **column**
- Same **diagonal**



So, we need to make sure no two queens share any of these.

Program Implementation:

We are using 4 methods to implement this program.

Method Overview

1. **SolveNQueen:** The main method that initiates the solution process.

2. **PrintBoard:** A method that prints the solution board with queens placed.

3. **Solve:** A recursive method that uses backtracking to place queens on the board.

4. **IsSafe:** A method that checks if a queen can be placed at a given position without being attacked.

How They Work Together

1. **SolveNQueen** calls **Solve** to start the backtracking process.

2. **Solve** uses **IsSafe** to check if a queen can be placed at a given position.

3. If a safe position is found, **Solve** places the queen and recursively calls itself for the next row.

4. If a solution is found, Solve calls **PrintBoard** to display the solution.

1. **SolveNQueen:**

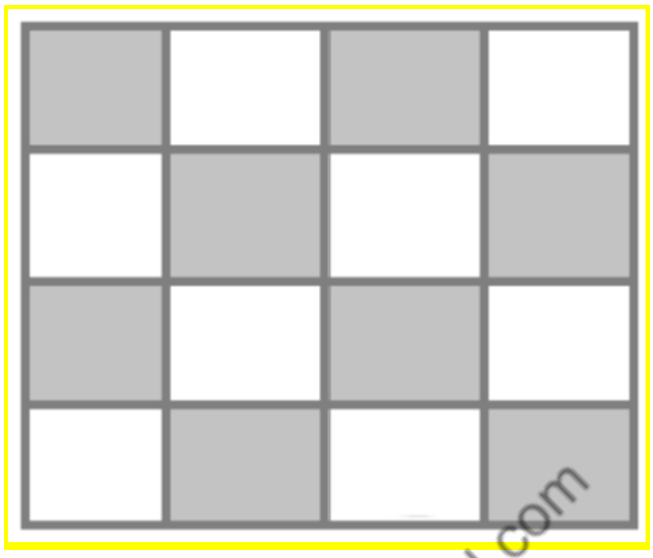
```
public static void solveNQueens(int n) {
    int[][] board = new int[n][n]; // Initialize an n x n board
    solve(0, board, n); // Start from row 0
}
```

2. **Solve Method:**

```
// Recursive function to place queens
private static void solve(int row, int[][] board, int n) {
    if (row == n) {
            solutionCount++; // Increment solution number
            System.out.println("Solution " + solutionCount + ":");
        printBoard(board, n); // All queens placed, print the solution
        return;
    }

    // Try placing queen in each column of current row
    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col, n)) {
            board[row][col] = 1;         // Place queen
            solve(row + 1, board, n);    // Recur to next row
            board[row][col] = 0;         // Backtrack
        }
    }
}
```

Solve Method Logic:



Step 1: Start at row 0

Try to place a queen in **each column of row 0**:

- Is (0, 0) safe? Yes → place queen
- Then go to next row (row = 1)

Step 2: Row 1

Try each column of **row 1**:

- (1, 0)? Check → blocked (same column as queen at 0,0) → skip
- (1, 1)? Check → diagonal attack → skip
- (1, 2)? Safe? Yes → place queen

Then go to row 2.

Step 3: Row 2

Try to place a queen in row 2...

- All columns may be unsafe

Step 4: Backtrack

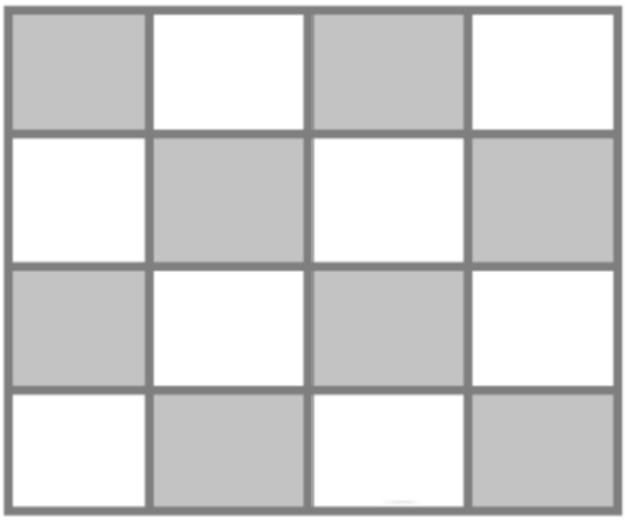
Go back to row 1:

- Remove the queen from (1, 2) → this is **backtracking**
- Try the next column → (1, 3)? Safe? Yes → place queen
- Continue to row 2

Step 5: Keep Going Until a Solution is Found

- Keep going like this until you reach row == n (you placed all queens safely).
- Then print the board.
- After that, you **continue backtracking** to explore other possible arrangements.

3. **isSafe Method:**



```
    // Function to check if it's safe to place a queen at (row, col)
    private static boolean isSafe(int[][] board, int row, int col, int n) {
        // Check column
        for (int i = 0; i < row; i++)
            if (board[i][col] == 1)
                return false;

        // Check upper-left diagonal
        for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
            if (board[i][j] == 1)
                return false;

        // Check upper-right diagonal
        for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)
            if (board[i][j] == 1)
                return false;

        return true; // No conflict, it's safe
    }
```

4. PrintBoard Method:

```
    // Function to print the board
    private static void printBoard(int[][] board, int n) {


        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(board[i][j] == 1 ? "Q " : ". ");

            }
            System.out.println();

        }
        System.out.println();


    }
```


5. Main Method:

```
    // Main function
    public static void main(String[] args) {
        int n = 8; // You can change N to 8, 5, etc.
        solveNQueens(n);
    }
}
```

Final Output :

```
Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .
Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .
```