LP II Artificial Intelligence

Practical No 4

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

Solution:

What is Branch and Bound:

Branch and Bound is an optimization technique used to efficiently solve problems like the N-Queen problem.

In the context of the N-Queen problem, it helps reduce the number of unnecessary recursive calls by eliminating infeasible solutions early **(bounding),** while exploring the possible placements of queens **(branching).**

N Queen Problem:

The problem is to place **N queens** on an **N×N chessboard** such that **no two queens attack each other** (i.e., no two queens share the same row, column, or diagonal).

So, the branch and bound technique avoid placing a queen in positions that are already known to be unsafe due to previously placed queens.

1. **Branching:**

Place a queen in a row and then move to the next row (recursively).

2. **Bounding:**

Before placing a queen, **check**:

- Is this column free?
- Is this main diagonal(upper left to bottom right) free?
- Is this anti-diagonal(upper right to bottom left) free?

If all are free → place queen and mark them as used.

Otherwise → **prune** this branch (don't continue down this path).

Finding Main Diagonal and Anti-Diagonal:

1. **Main Diagonal:**

1. Main diagonal cells all have the same `row - col` value
2. We add `N - 1` to avoid negative indices (since `row - col` can be negative)
3. This maps all possible diagonals to a valid index in the array

Example for N = 4:

Cell	row	col	row - col	d1 = row - col + 3
(0, 0)	0	0	0	3
(0, 1)	0	1	-1	2
(1, 0)	1	0	1	4
(3, 0)	3	0	3	6

2. Anti-diagonal :

Anti-diagonal cells all have the same `row + col` value `row + col` is already non-negative

Example for N = 4:

Cell	row	col	row + col	d2
(0, 3)	0	3	3	3
(1, 2)	1	2	3	3
(2, 1)	2	1	3	3
(3, 0)	3	0	3	3

d1 represents main diagonal and d2 represents antidiagonal value

So finally,

3	2	1	0
4	3	2	1
5	4	3	2
6	5	4	3

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

Program Implementation:

1. **Declare class and initialize data structures:**

```
public class NQueenBranchBound {

    public static void main(String[] args) {
        int N = 4; // Change N to solve for different board sizes

        // Declare and initialize data structures
        int[][] board = new int[N][N];
        boolean[] cols = new boolean[N];
        boolean[] diag1 = new boolean[2 * N - 1]; // for row - col + N - 1
        boolean[] diag2 = new boolean[2 * N - 1]; // for row + col

        // Solve the N-Queen problem starting from row 0
        solve(0, N, board, cols, diag1, diag2);
    }
```


2. Solve Method:

```java
// Method to solve the N-Queen problem using Branch and Bound
public static void solve(int row, int N, int[][] board, boolean[] cols, boolean[] diag1, boolean[] diag2) {
    if (row == N) {
        printSolution(N, board);
        return;
    }

    for (int col = 0; col < N; col++) {
        int d1 = row - col + N - 1; // main diagonal
        int d2 = row + col;         // anti-diagonal

        if (!cols[col] && !diag1[d1] && !diag2[d2]) {
            board[row][col] = 1;
            cols[col] = diag1[d1] = diag2[d2] = true;

            solve(row + 1, N, board, cols, diag1, diag2);

            board[row][col] = 0;
            cols[col] = diag1[d1] = diag2[d2] = false;
        }
    }
}
```

3. Print Solution:

```
    // Method to print the board with queens placed
    public static void printSolution(int N, int[][] board) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i][j] == 1)
                    System.out.print("Q ");
                else
                    System.out.print(". ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```