**Practical – 05**

**Aim :**

Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen‘s matrix.

**Program:**

import java.util.\*;

class NQueens1 {

static int N = 4;

// ld is an array where its indices indicate row-col+N-1

// (N-1) is for shifting the difference to store

// negative indices

static int[] ld = new int[30];

// rd is an array where its indices indicate row+col

// and used to check whether a queen can be placed on

// right diagonal or not

static int[] rd = new int[30];

// Column array where its indices indicates column and

// used to check whether a queen can be placed in that

// row or not

static int[] cl = new int[30];

// A utility function to print solution

static void printSolution(int board[][])

{

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

System.out.printf(" %d ", board[i][j]);

System.out.printf("\n");

}

}

// A recursive utility function to solve N

// Queen problem

static boolean solveNQUtil(int board[][], 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]

// To check if a queen can be placed on

// board[row][col].We just need to check

// ld[row-col+n-1] and rd[row+coln] where

// ld and rd are for left and right

// diagonal respectively

if ((ld[i - col + N - 1] != 1

&&rd[i + col] != 1)

&& cl[i] != 1) {

// Place this queen in board[i][col]

board[i][col] = 1;

ld[i - col + N - 1] = rd[i + col] = cl[i]

= 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

ld[i - col + N - 1] = rd[i + col] = cl[i]

= 0;

}

}

// 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.

static boolean solveNQ()

{

int board[][] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

System.out.printf("Solution does not exist");

return false;

}

printSolution(board);

return true; }

// Driver Code

public static void main(String[] args)

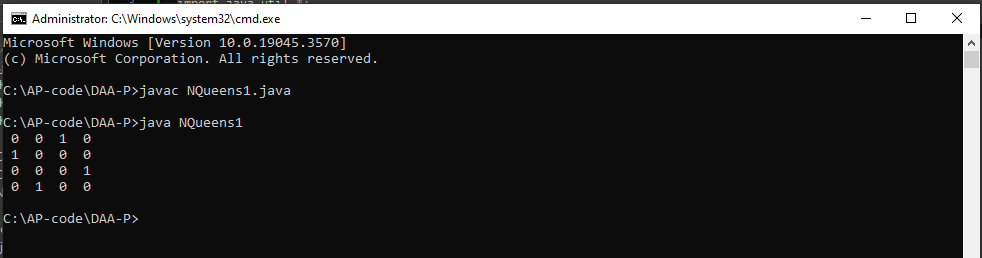
{ solveNQ();

} }

/\* Time Complexity: O(N!)

Auxiliary Space: O(N^2) \*/

**Output:**

****