**ASSIGNMENT NO.4**

import java.io.\*;

import java.util.Arrays;

class B4 {

public static void main(String args[]) {

int N = 8;

NQBranchAndBond NQBaB = new NQBranchAndBond(N);

NQBaB.solveNQ();

NQBacktracking NQBt = new NQBacktracking(N);

NQBt.solveNQ();

}

}

class NQBranchAndBond {

private int N;

NQBranchAndBond(int N) {

this.N = N;

}

void printSolution(int board[][]) {

System.out.println("N Queen Branch And Bound Solution:");

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

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

System.out.printf("%2d ", board[i][j]);

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

}

}

static boolean isSafe (

int row, int col,

int slashCode[][],

int backslashCode[][],

boolean rowLookup[],

boolean slashCodeLookup[],

boolean backslashCodeLookup[]

) {

return !(

slashCodeLookup[slashCode[row][col]] ||

backslashCodeLookup[backslashCode[row][col]] ||

rowLookup[row]

);

}

boolean solveNQUtil(

int board[][], int col, int slashCode[][],

int backslashCode[][], boolean rowLookup[],

boolean slashCodeLookup[], boolean backslashCodeLookup[]

) {

if (col >= N)

return true;

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

if (isSafe(

i, col, slashCode,

backslashCode, rowLookup,

slashCodeLookup, backslashCodeLookup

)) {

board[i][col] = 1;

rowLookup[i] = true;

slashCodeLookup[slashCode[i][col]] = true;

backslashCodeLookup[backslashCode[i][col]] = true;

if (solveNQUtil(

board, col + 1, slashCode,

backslashCode, rowLookup,

slashCodeLookup,

backslashCodeLookup

))

return true;

board[i][col] = 0;

rowLookup[i] = false;

slashCodeLookup[slashCode[i][col]] = false;

backslashCodeLookup[backslashCode[i][col]] = false;

}

}

return false;

}

boolean solveNQ() {

int board[][] = new int[N][N];

int slashCode[][] = new int[N][N];

int backslashCode[][] = new int[N][N];

boolean[] rowLookup = new boolean[N];

boolean slashCodeLookup[] = new boolean[2 \* N - 1];

boolean backslashCodeLookup[] = new boolean[2 \* N - 1];

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

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

slashCode[r][c] = r + c;

backslashCode[r][c] = r - c + N - 1;

}

if (solveNQUtil(

board, 0, slashCode,

backslashCode, rowLookup,

slashCodeLookup,

backslashCodeLookup

) == false) {

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

return false;

}

printSolution(board);

return true;

}

}

class NQBacktracking {

private int N;

NQBacktracking(int N){

this.N = N;

}

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

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

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

void printSolution(int board[][]) {

System.out.println("\n\nN Queen Backtracking Solution:");

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

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

System.out.printf("%2d ", board[i][j]);

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

}

}

boolean solveNQUtil(int board[][], int col) {

if (col >= N)

return true;

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

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

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

board[i][col] = 1;

ld[i - col + N - 1] =

rd[i + col] = cl[i] = 1;

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0;

ld[i - col + N - 1] =

rd[i + col] = cl[i] = 0;

}

}

return false;

}

boolean solveNQ() {

int board[][] = new int[N][N];

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

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

return false;

}

printSolution(board);

return true;

}

}

**OUTPUT:-**

N Queen Branch And Bound Solution:

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0

N Queen Backtracking Solution:

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0