CLASS – HILL CLIMBING :RANDOM RESTART ALGORITHM

```
import java.util.Random;
import java.util.Scanner;


public class HillClimbingRestart {
private static int n;
private static int stepsClimbedAfterLastRestart = 0;
private static int stepsClimbed = 0;
private static int heuristic = 0;
private static int randomRestarts = 0;


// Method to create a new random board
public static NQueen[] generateBoard() {
NQueen[] startBoard = new NQueen[n];
Random rndm = new Random();
for (int i = 0; i < n; i++) {
startBoard[i] = new NQueen(rndm.nextInt(n), i);
}
return startBoard;
}


// Method to print the Current State
private static void printState (NQueen[] state) {
//Creating temporary board from the present board
int[][] tempBoard = new int[n][n];
for (int i=0; i<n; i++) {
```

```
//Get the positions of Queen from the Present board and set those positions as 1 in temp board
tempBoard[state[i].getRow()][state[i].getColumn()]=1;
}
System.out.println();
for (int i=0; i<n; i++) {
for (int j= 0; j < n; j++) {
System.out.print(tempBoard[i][j] + " ");
}
System.out.println();
}
}


// Method to find Heuristics of a state
public static int findHeuristic(NQueen[] state) {
int heuristic = 0;
for (int i = 0; i < state.length; i++) {
for (int j = i + 1; j < state.length; j++) {
if (state[i].ifConflict(state[j])) {
heuristic++;
}
}
}
return heuristic;
}


// Method to get the next board with lower heuristic
public static NQueen[] nextBoard(NQueen[] presentBoard) {
```

```
NQueen[] nextBoard = new NQueen[n];
NQueen[] tmpBoard = new NQueen[n];
int presentHeuristic = findHeuristic(presentBoard);
int bestHeuristic = presentHeuristic;
int tempH;
for (int i = 0; i < n; i++) {
// Copy present board as best board and temp board
nextBoard[i] = new NQueen(presentBoard[i].getRow(), presentBoard[i].getColumn());
tmpBoard[i] = nextBoard[i];
}
// Iterate each column
for (int i = 0; i < n; i++) {
if (i > 0)
tmpBoard[i - 1] = new NQueen(presentBoard[i - 1].getRow(), presentBoard[i - 1].getColumn());
tmpBoard[i] = new NQueen(0, tmpBoard[i].getColumn());
// Iterate each row
for (int j = 0; j < n; j++) {
// Get the heuristic
tempH = findHeuristic(tmpBoard);
// Check if temp board better than best board
if (tempH < bestHeuristic) {
bestHeuristic = tempH;
// Copy the temp board as best board
for (int k = 0; k < n; k++) {
nextBoard[k] = new NQueen(tmpBoard[k].getRow(), tmpBoard[k].getColumn());
}
}
// Move the queen
```

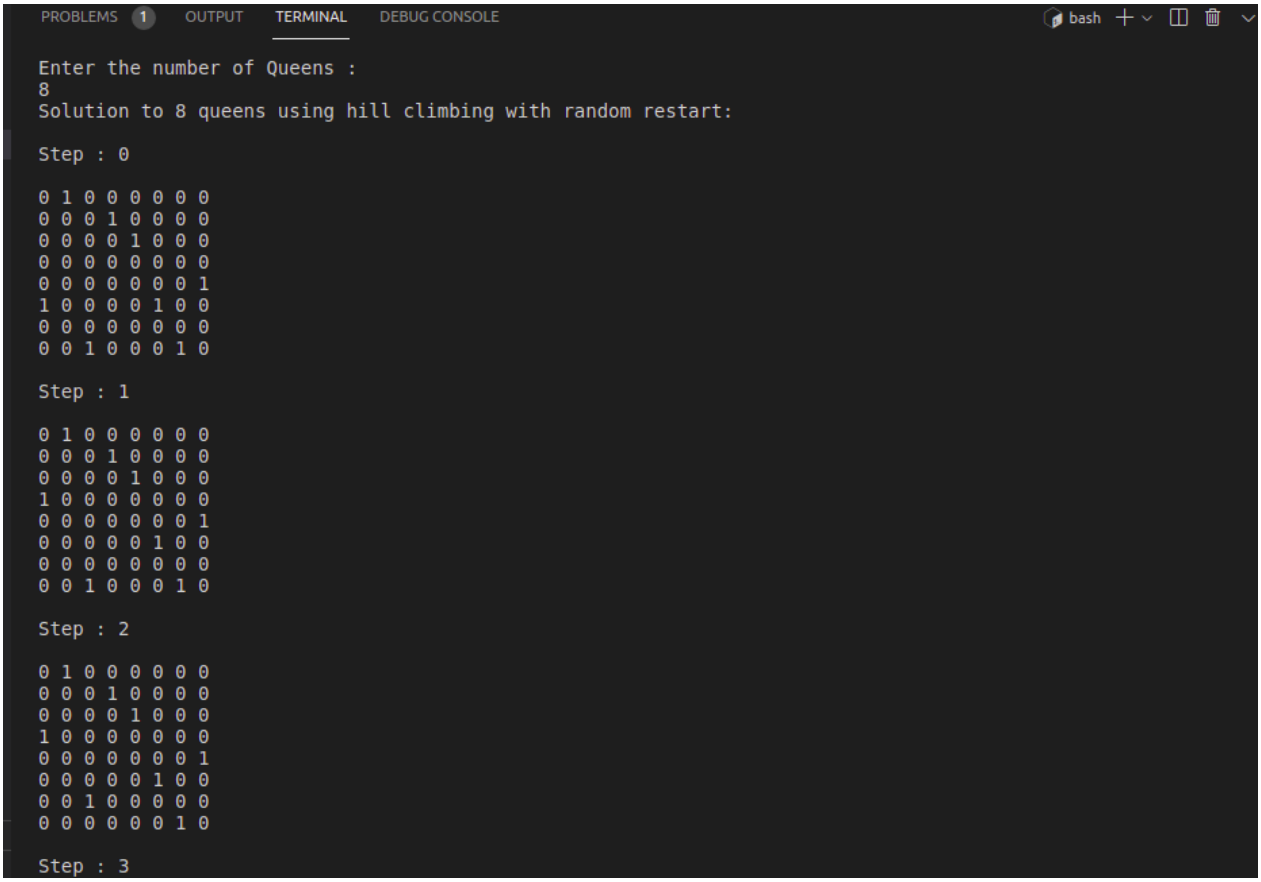
```
if (tmpBoard[i].getRow() != n - 1)
tmpBoard[i].move();
}
}
// Check whether the present bord and the best board found have same heuristic
// Then randomly generate new board and assign it to best board
if (bestHeuristic == presentHeuristic) {
randomRestarts++;
stepsClimbedAfterLastRestart = 0;
nextBoard = generateBoard();
heuristic = findHeuristic(nextBoard);
} else
heuristic = bestHeuristic;
stepsClimbed++;
stepsClimbedAfterLastRestart++;
return nextBoard;
}


public static void main(String[] args) {
int presentHeuristic;
Scanner s=new Scanner(System.in);
while (true){
System.out.println("Enter the number of Queens :");
n = s.nextInt();
if ( n == 2 || n ==3) {
System.out.println("No Solution possible for "+n+" Queens. Please enter another
number");
}
else
```

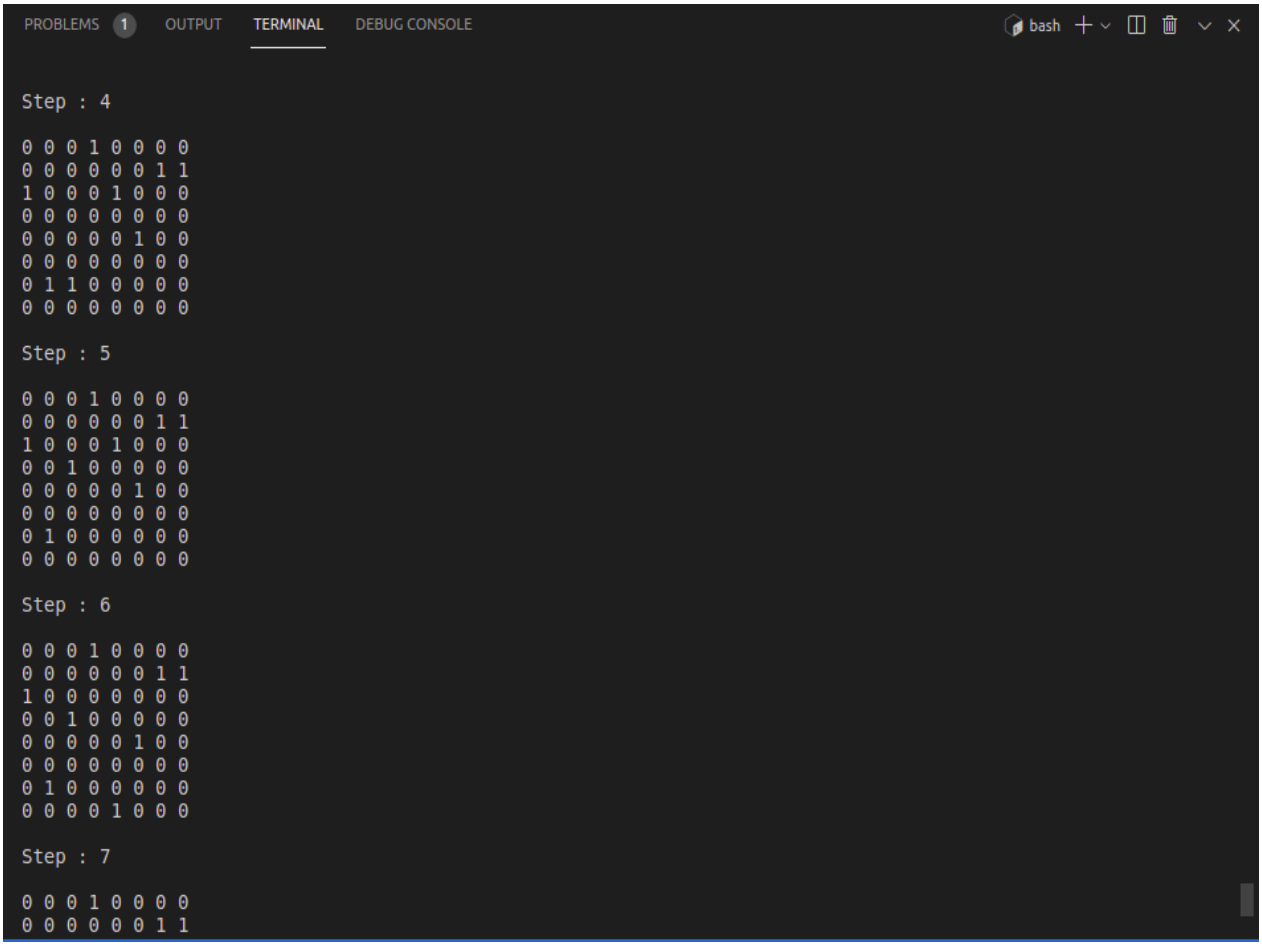
```
break;
}
System.out.println("Solution to "+n+" queens using hill climbing with random restart:");
//Creating the initial Board
NQueen[] presentBoard = generateBoard();
presentHeuristic = findHeuristic(presentBoard);
// test if the present board is the solution board
while (presentHeuristic != 0) {
// Get the next board
System.out.println("\nStep : "+stepsClimbed);
printState(presentBoard);


presentBoard = nextBoard(presentBoard);
presentHeuristic = heuristic;
}
//Printing the solution\
System.out.println("\nFinal Solution");
printState(presentBoard);
System.out.println("\n"+presentHeuristic+"\nTotal number of Steps Climbed: " +
stepsClimbed);
System.out.println("Number of random restarts: " + randomRestarts);
System.out.println("Steps Climbed after last restart: " +
stepsClimbedAfterLastRestart);
}
}
```

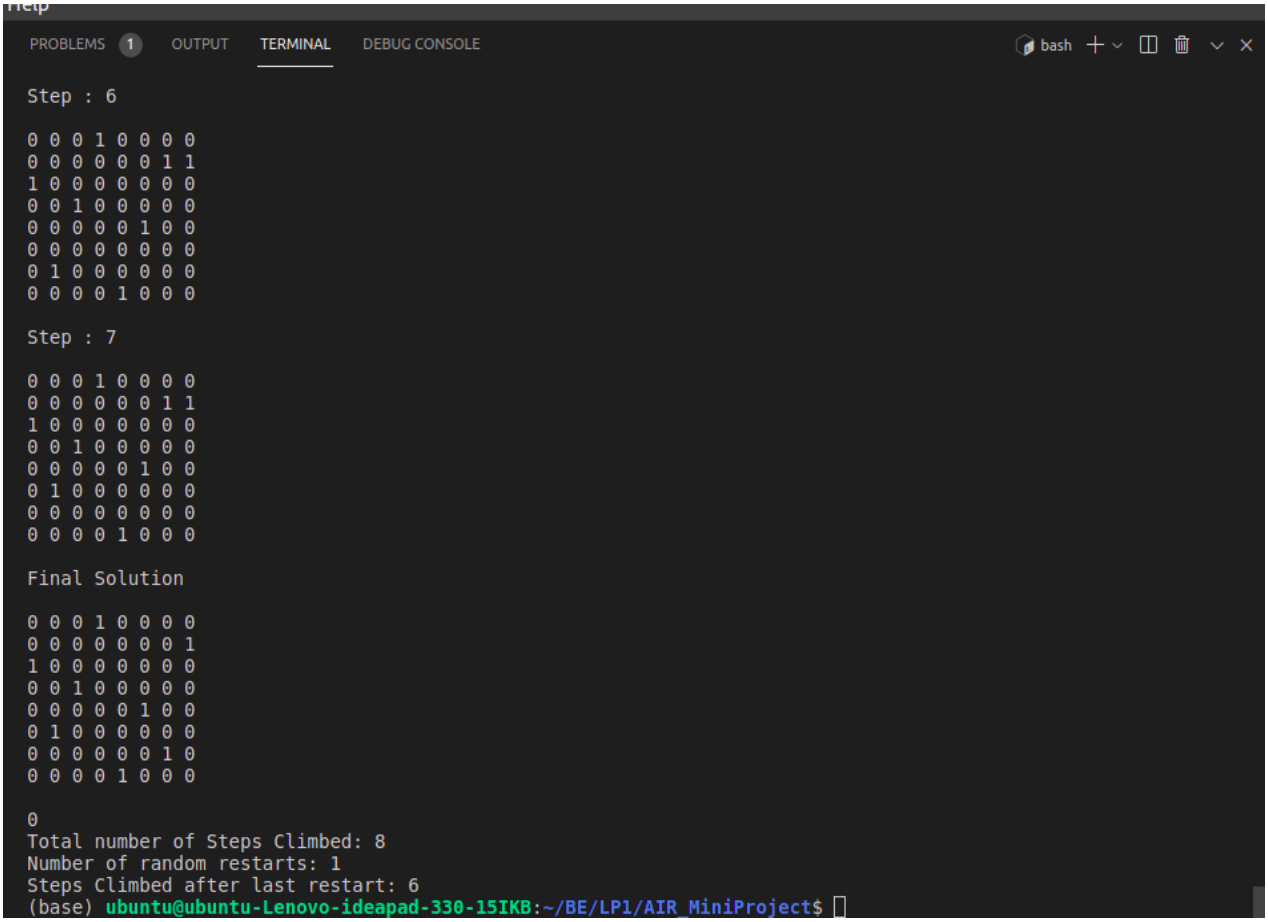
CLASS – NQUEEN

```
public class NQueen {
private int row;
private int column;


public NQueen(int row, int column) {
this.row = row;
this.column = column;
}
public void move() {
row++;
}
public boolean ifConflict(NQueen q) {
// Check rows and columns
if (row == q.getRow() || column == q.getColumn())
return true;
// Check diagonals
else if (Math.abs(column - q.getColumn()) == Math.abs(row - q.getRow()))
return true;
return false;
}
public int getRow() {
return row;
}
public int getColumn() {
return column;
}
}
```

OUTPUT – SCREENSHOTS




PROBLEMS 1
OUTPUT
TERMINAL
DEBUG CONSOLE
bash
Step : 4
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
Step : 5
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Step : 6
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
Step : 7
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1


PROBLEMS 1
OUTPUT
TERMINAL
DEBUG CONSOLE
bash
Step : 6
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
Step : 7
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
Final Solution
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0
Total number of Steps Climbed: 8
Number of random restarts: 1
Steps Climbed after last restart: 6
(base) ubuntu@ubuntu-Lenovo-ideapad-330-15IKB:~/BE/LP1/AIR_MiniProject$