AIR MiniProject

Roll no- 41232,41239

Problem Statement : Nqueen Problem using Hill Climbing Algorithm

Abstract :

1.Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

2.Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

3.In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

4.Nqueens can be solved by different approaches like BackTracking,Bit-Masking,Parallel Implementation of Bit-Masking,Hill-Climbing etc.

5.In this miniproject we are solving it using Hill-Climbing approach.

H/W and S/W Requirement : Java,Eclipse,JDK,64 bit OS, 8 GB RAM, 500 GB HDD,Monitor,Keyboard.

Objective : To understand usage of Hill Climbing algorithms to solve N Queen’s problem.

Introduction:

1. Hill Climbing with random restart is a local search algorithm. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution.

2.If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

3.The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.

4.The hill-climbing algorithms often fail to find a goal when one exists because they can get stuck on local maxima.

5.Random-restart can be used to solve the problem of local maxima, as it conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

Scope :

1. In this we are finding out the board state for given number of queens.

2.The scope of this is to check teh performance of the hill climbing algorithm for n-queens problem in comparison to Parallel-Implementation and backtracking.

System Architecture :

The system consists of two Classes:

1. HillClimbingRandomRestart: The main class which has the hill climbing algorithm with random restart and the main method of the program.
2. NQueen: The NQueen Class is used to represent the N-queens state.

NQueen Class:

The NQueen class is used to represent the N-queens state. It has the following components:

1. row: Current row of the queen
2. column: Current column of the queen The class has getter methods for both components and a parameterized constructor. This class has following methods:
3. ifConflict(NQueen q): This method checks if the queen has any conflicts or not and returns true if there are conflicts otherwise returns false.
4. move(): Move the queen one row down

The HillClimbingRandomRestart class is the main class which accepts the input from user, contains the Hill climbing with random restart algorithm logic and prints the desired output to the console. This class has the following methods:

1. Main() : This accepts the input from user and checks if the input is valid, then generate an initial board randomly and then perform hill climbing. Finally print the number of steps climbed and total number of random restarts used.
2. generateBoard(): This method generates a board and fills the n-queens randomly. This is used at the beginning to generate the initial board and is called for each random restart thereafter.
3. printState(Nqueen[] State): This is a helper method to print the current configuration.
4. findHeuristic(NQueen[] State): This method finds the number of conflicts of the queen.
5. nextBoard (NQueen[] presentBoard): This method finds the next configuration i.e. the step to be climbed if there is any.

Test Cases :

For n=4 queens

Enter the number of Queens :
4
Solution to 4 queens using hill climbing with random restart:

Heuristic val3

0 0 1 1
0 0 0 0
0 0 0 0
1 1 0 0

Heuristic val1

0 0 1 1
1 0 0 0
0 0 0 0
0 1 0 0
Final Configuration

0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

Expected Output – Queens arranged such that no attacking takes place

For n=8 queens -

Enter the number of Queens :
8
Solution to 8 queens using hill climbing with random restart:

Heuristic val4

0 0 1 0 1 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 1 0
0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 0

Heuristic val2

0 0 1 0 1 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 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0

Heuristic val13

0 0 0 0 0 0 0 0
0 1 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 0 0 0 0 0
0 0 1 0 1 0 0 0
1 0 0 1 0 1 1 0
0 0 0 0 0 0 0 0

Heuristic val7

0 0 0 1 0 0 0 0
0 1 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 0 0 0 0 0
0 0 1 0 1 0 0 0
1 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0

Heuristic val4

0 0 0 1 0 0 0 0
0 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
0 0 1 0 1 0 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

Heuristic val3

0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 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

Heuristic val2

0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0

Final Configuration

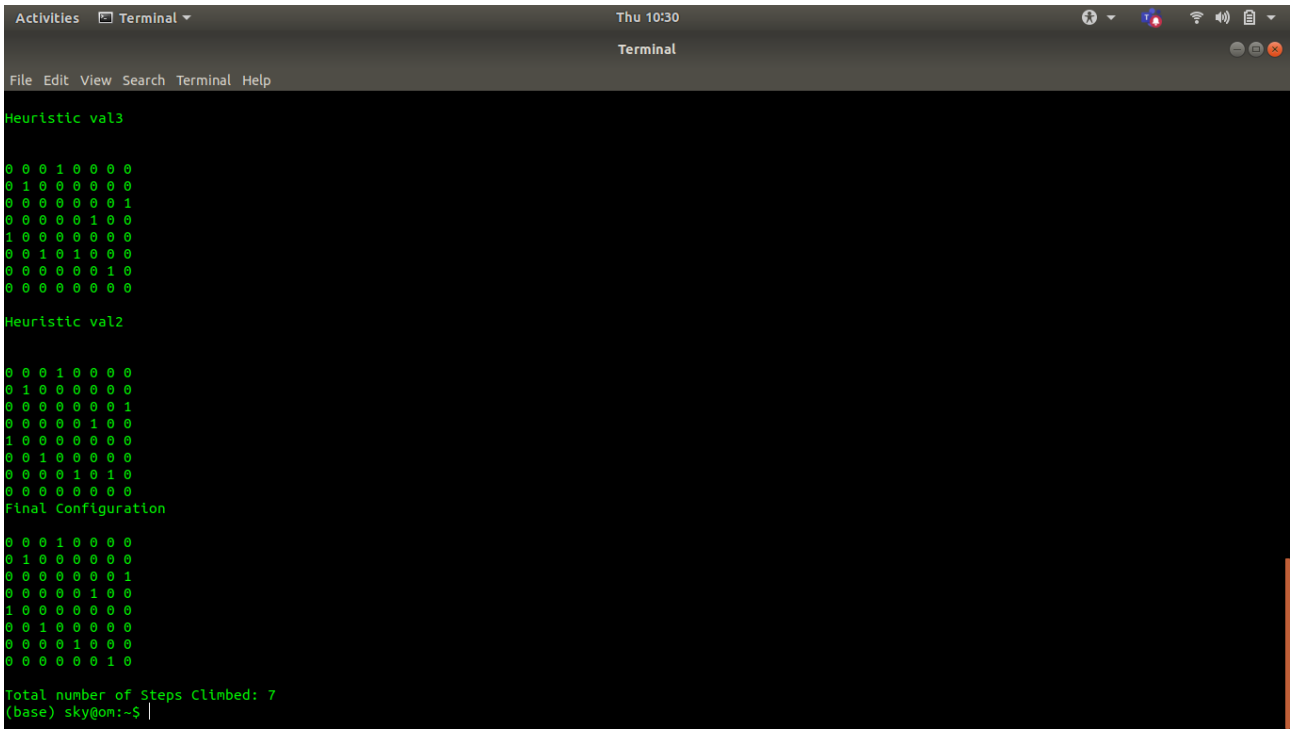
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0

Total number of Steps Climbed: 7

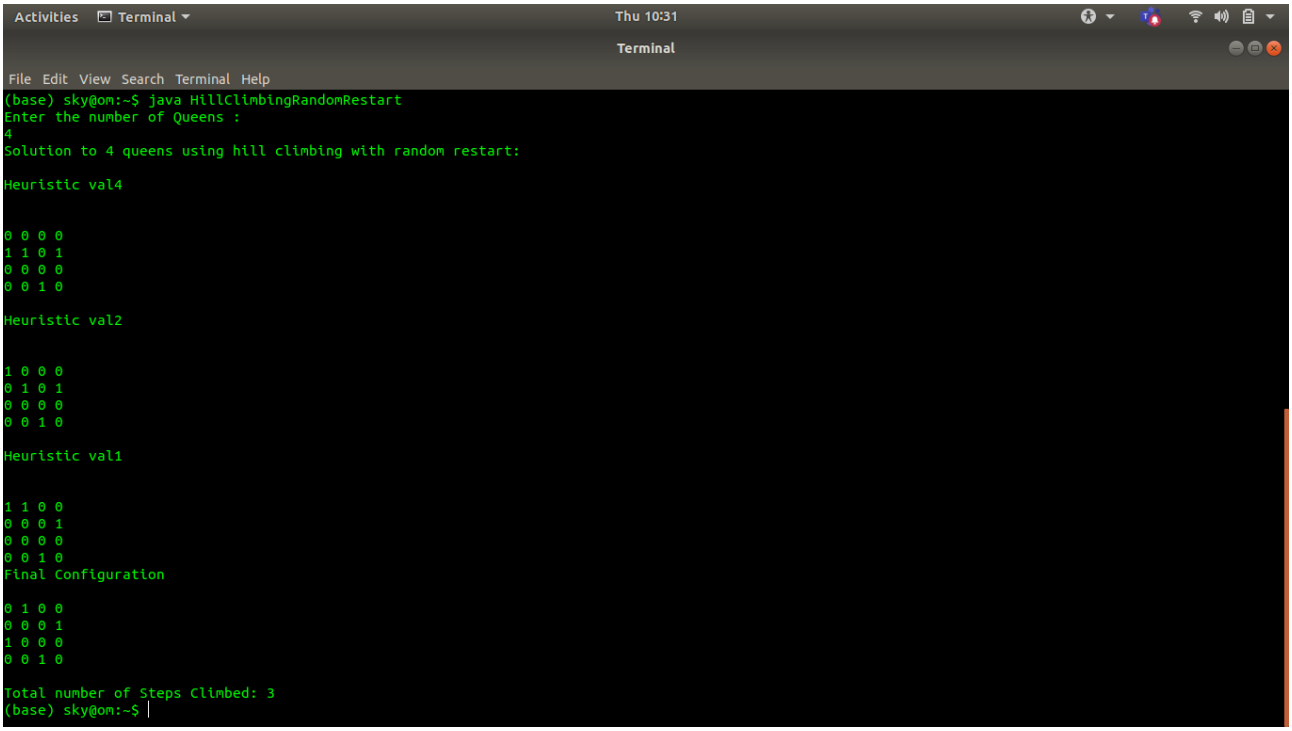
Expected Output – Queens arranged such that no attacking takes place

Results :

For n = 8



For n = 4



Conclusion :

We successfully implmented the Hill Climbing approach for N-Queens Problem.Using the Hill-Climbing the different steps needed to reach teh solution algon with heuristic values are printed.

On Compairing with the other approaches we can conclude that

Parallel Implementation > Hill-Climbing Implemetation > Backtracking

References :

[1] www.javatpoint.com for Algorithm details

[2] www.geeksforgeeks.com for Heuristic values study

[3] www.cs.cmu.edu for Hill-Climbing algorithm N-Queens

Code :

HillClimbingRandomRestart.java

```
//Program to implement Hill Climbing with random restart to solve N-queens problem
import java.util.Scanner;
import java.util.Random;
public class HillClimbingRandomRestart {
    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
           // printState(presentBoard);
            presentBoard = nextBoard(presentBoard);
            presentHeuristic  = heuristic;
        }
        //Printing the 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);
    }
}
```

Nqueen.java

```
//Class for N-queens Problem
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;
    }
}
```