**ASSIGNMENT 8**

**Aim:** Use heuristic search technique to implement Hill Climbing Algorithm.

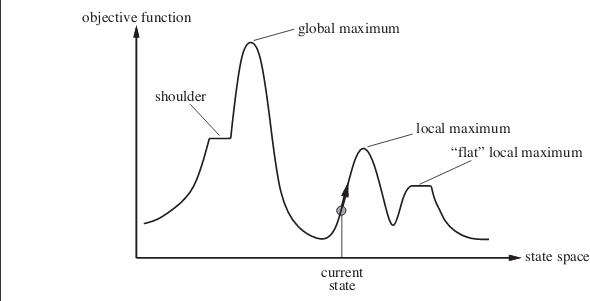
**Software and Hardware Requirements:** Netbeans IDE.

**Objective:** To solve 8 queens problem using Hill Climbing Algorithm.

### Theory:

Hill Climbing is a technique to solve certain optimization problems. Here, we start with a suboptional solution and the solution is improved repeatedly until some condition is maximized.

### Diagram:



The idea of starting with a sub-optimal solution is compared to starting from the base of the hill, improving the solution is compared to walking up the hill and finally maximizing some condition is compared to reaching the top of the hill.

Hence, the hill climbing technique can be considered as the following phases:

* + Constructing a sub-optimal solution obeying the constraints of the problem.
  + Improving the solution step by step.
  + Improving the solution until no more improvement is possible.

A heuristic is a technique to solve problem faster than classic methods or to find an approximate solution when classic methods cannot heuristic are categorized into two categories:

1. Direct Heuristic Search Technique.
2. Weak Heuristic Search Technique: The hill climbing algorithm implementation falls under thios category.

Heuristic Search :- Types of Hill Climbing in AI

* + Simple Hill Climbing
  + Steepest Ascent Hill Climbing
  + Stochastic Hill Climbing

### Input:

Queens are placed on board depending upon the generated random number. Function for placing queen:

public void placeQueens()

{

queenPositions.generateQueens(); for(int i=0;i<board.length;i++)

{

Board[queenPositions[i]][i]=1;

}

}

queenPositions[i] is the random number generated.

Therefore, for the following random numbers the queen will be placed on board like this:

|  |  |
| --- | --- |
| Random Numbers Generated: | Corresponding Board: |
| 1 | 00000000 |
| 3 | 00001011 |
| 5 | 00000000 |
| 7 | 01000000 |
| 1 | 00000000 |
| 5 | 00100100 |
| 1 | 00000000 |
| 1 | 10010000 |

**Code:**

package queensboard;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

public class QueensBoard {

public static int TOTAL\_QUEENS = 8;

private int[][] board;

private int[] queenPositions;

public static void main(String[] args) {

boolean climb = true;

int climbCount = 0;

QueensBoard board = new QueensBoard(

new int[TOTAL\_QUEENS][TOTAL\_QUEENS], new int[8]);

board.placeQueens();

System.out.println("Trial #: " + (climbCount+1));

System.out.println("Original board:");

board.printBoard();

System.out.println("# pairs of queens attacking each other: "

+ board.h() + "\n");

// score to be compared against

int localMin = board.h();

boolean best = false;

// array to store best queen positions by row (array index is column)

int[] bestQueenPositions = new int[8];

// iterate through each column

for (int j = 0; j < board.board.length; j++) {

System.out.println("Iterating through COLUMN " + j + ":");

best = false;

// iterate through each row

for (int i = 0; i < board.board.length; i++) {

// skip score calculated by original board

if (i != board.queenPositions[j]) {

// move queen

board.moveQueen(i, j);

board.printBoard();

System.out.println();

// calculate score, if best seen then store queen position

if (board.h() < localMin) {

best = true;

localMin = board.h();

System.out.println("# pairs of queens attacking each other(board

below):" + localMin);

bestQueenPositions[j] = i;

}

// reset to original queen position

board.resetQueen(i, j);

}

}

// change 2 back to 1

board.resetBoard(j);

if (best) {

// if a best score was found, place queen in this position

board.placeBestQueen(j, bestQueenPositions[j]);

System.out.println("Best board found this iteration: ");

board.printBoard();

System.out

.println("# pairs of queens attacking each other: "

+ board.h() + "\n");

} else {

System.out.println("No better board found.");

board.printBoard();

System.out

.println("# pairs of queens attacking each other: "

+ board.h() + "\n");

}

}

// if score = 0, hill climbing has solved problem

if (board.h() == 0)

climb = false;

climbCount++;

// only 5 restarts

if (climbCount == 6) {

climb = false;

}

System.out.println("Done in " + (climbCount-1) + " restarts.");

}

public QueensBoard(int[][] board, int[] positions) {

this.board = board;

this.queenPositions = positions;

}

private int[] generateQueens() {

List<Integer> startState = new ArrayList<Integer>();

Scanner in = new Scanner(System.in);

int value;

System.out.println("Enter 8 row nos (0-7) of queens to be placed"+"\n

"+"Queens will be placed in every column at the row no specified. ");

for (int i = 0; i < TOTAL\_QUEENS; i++) {

value = in.nextInt();

startState.add(value);

}

int[] Positions = new int[TOTAL\_QUEENS];

for (int i = 0; i < startState.size(); i++) {

Positions[i] = startState.get(i);

}

return Positions;

}

public void placeQueens() {

queenPositions = generateQueens();

for (int i = 0; i < board.length; i++) {

board[queenPositions[i]][i] = 1;

}

}

public void placeQueens2()

{

for (int i = 0; i < board.length; i++) {

board[queenPositions[i]][i] = 1;

}

}

public int h() {

int totalPairs = 0;

// checking rows

for (int i = 0; i < board.length; i++) {

ArrayList<Boolean> pairs = new ArrayList<Boolean>();

for (int j = 0; j < board[i].length; j++) {

if (board[i][j] == 1) {

pairs.add(true);

}

}

if (pairs.size() != 0)

totalPairs = totalPairs + (pairs.size() - 1);

}

// check diagonal from top left

int rows = board.length;

int cols = board.length;

int maxSum = rows + cols - 2;

for (int sum = 0; sum <= maxSum; sum++) {

ArrayList<Boolean> pairs = new ArrayList<Boolean>();

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

for (int j = 0; j < cols; j++) {

if (i + j - sum == 0) {

if (board[i][j] == 1) {

pairs.add(true);

}

}

}

}

if (pairs.size() != 0)

totalPairs = totalPairs + (pairs.size() - 1);

}

int pairs = checkMirrorDiagonal();

return totalPairs + pairs;

}

private int checkMirrorDiagonal() {

int[] b1 = { board[7][0] };

int[] b2 = { board[7][1], board[6][0] };

int[] b3 = { board[7][2], board[6][1], board[5][0] };

int[] b4 = { board[7][3], board[6][2], board[5][1], board[4][0] };

int[] b5 = { board[7][4], board[6][3], board[5][2], board[4][1],

board[3][0] };

int[] b6 = { board[7][5], board[6][4], board[5][3], board[4][2],

board[3][1], board[2][0] };

int[] b7 = { board[7][6], board[6][5], board[5][4], board[4][3],

board[3][2], board[2][1], board[1][0] };

int[] b8 = { board[7][7], board[6][6], board[5][5], board[4][4],

board[3][3], board[2][2], board[1][1], board[0][0] };

int[] b9 = { board[6][7], board[5][6], board[4][5], board[3][4],

board[2][3], board[1][2], board[0][1] };

int[] b10 = { board[5][7], board[4][6], board[3][5], board[2][4],

board[1][3], board[0][2] };

int[] b11 = { board[4][7], board[3][6], board[2][5], board[1][4],

board[0][3] };

int[] b12 = { board[3][7], board[2][6], board[1][5], board[0][4] };

int[] b13 = { board[2][7], board[1][6], board[0][5] };

int[] b14 = { board[1][7], board[0][6] };

int[] b15 = { board[0][7] };

int totalPairs = 0;

totalPairs += checkMirrorHorizontal(b1);

totalPairs += checkMirrorHorizontal(b2);

totalPairs += checkMirrorHorizontal(b3);

totalPairs += checkMirrorHorizontal(b4);

totalPairs += checkMirrorHorizontal(b5);

totalPairs += checkMirrorHorizontal(b6);

totalPairs += checkMirrorHorizontal(b7);

totalPairs += checkMirrorHorizontal(b8);

totalPairs += checkMirrorHorizontal(b9);

totalPairs += checkMirrorHorizontal(b10);

totalPairs += checkMirrorHorizontal(b11);

totalPairs += checkMirrorHorizontal(b12);

totalPairs += checkMirrorHorizontal(b13);

totalPairs += checkMirrorHorizontal(b14);

totalPairs += checkMirrorHorizontal(b15);

return totalPairs;

}

public void moveQueen(int row, int col) {

// original queen will become a 2 and act as a marker

board[queenPositions[col]][col] = 2;

board[row][col] = 1;

}

private int checkMirrorHorizontal(int[] b) {

int totalPairs = 0;

ArrayList<Boolean> pairs = new ArrayList<Boolean>();

for (int i = 0; i < b.length; i++) {

if (b[i] == 1)

pairs.add(true);

}

if (pairs.size() != 0)

totalPairs = (pairs.size() - 1);

return totalPairs;

}

public void resetQueen(int row, int col) {

if (board[row][col] == 1)

board[row][col] = 0;

}

public void resetBoard(int col) {

for (int i = 0; i < board.length; i++) {

if (board[i][col] == 2)

board[i][col] = 1;

}

}

public void placeBestQueen(int col, int queenPos) {

for (int i = 0; i < board.length; i++) {

if (board[i][col] == 1)

board[i][col] = 2;

}

board[queenPos][col] = 1;

for (int i = 0; i < board.length; i++) {

if (board[i][col] == 2)

board[i][col] = 0;

}

}

public void printBoard() {

for (int i = 0; i < board.length; i++) {

for (int j = 0; j < board[i].length; j++) {

System.out.print(board[i][j] + " ");

}

System.out.println();

}

}

}

/\*

Output:

Enter 8 row nos (0-7) of queens to be placed

Queens will be placed in every column at the row no specified.

1 5 7 0 6 2 4 3

Trial #: 1

Original board:

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 0:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 1:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 2:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 3:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 4:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 5:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 6:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Iterating through COLUMN 7:

No better board found.

0 0 0 1 0 0 0 0

1 0 0 0 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 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

# pairs of queens attacking each other: 3

Done in 0 restarts.

\*/

**CONCLUSION:**

Hence, implementation of Hill Climbing Algorithm using Heuristic Search Technique is implemented.