Project 2

ITCS-6150 - Intelligent Systems

Topic : Solving n-queens problem using Hill Climbing Search and its variants

Submitted by

Snehal Kulkarni (801147615)
Dheeraj Mirashi (801151305)
Adwait More (801134139)

Date : 27th October, 2019

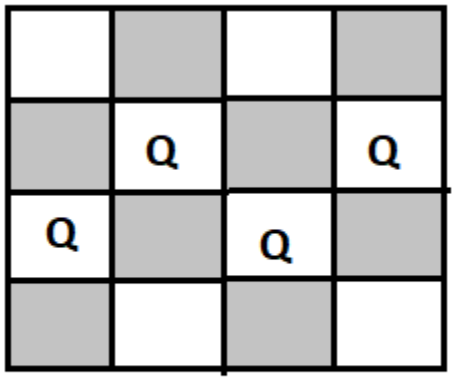
University of North Carolina at Charlotte

1) N-Queens problem formulation:

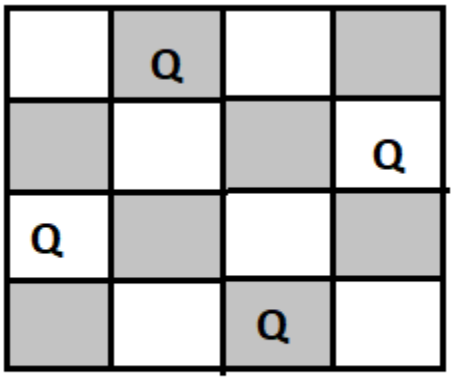
N-Queens puzzle is problem where n queens are put on n*n game board such that no two queens attack each other, directly or indirectly, meaning, no two queens are placed in the same row, in the same column and in the same diagonal. We say, the solution is found if we get such gameboard. We have implemented Hill Climbing search algorithm in Java programming language to solve this puzzle. We accept a board with initial state and solve the puzzle to get goal state as shown in the diagram:

2) Terminologies of the algorithm:

1) **State:** Any placement of 0 to n queens on the board is called as state.
2) **Initial State:** This is an initial state where all n queens are placed on the board randomly.This is called as complete-state formulation.
3) **Goal State:** A board with n queen having no two queens in the same row or column or diagonal.
4) **Action:** Adding a queen to any empty square.
5) **Transition Model:** Returns a board with a queen added to the specified square.



Initial State



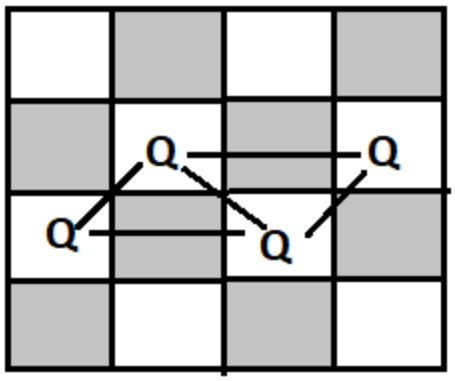
Goal State

3) Hill Climbing Search Algorithm:

This algorithm accepts the board with initial state and then finds the successors at each step in order to get the goal state. New state is formed by moving each queen up or down in the same column or it can be moved left or right in the same row. The best successor is selected with the lowest value of heuristic function. If all the successors have the same h value then the algorithm selects first successor to expand further. Successors can be calculated with formula n*(n-1).

4) Heuristic Function:

This function calculates heuristic value which is nothing but the number of queens attacking each other at the given state of board. Heuristic value is calculated for each successor and one with lowest h value is expanded to get further successors. For instance, h value for the initial state is as shown below.



h=5

This process stops when the goal state having h=0 is achieved.

5)Problems in Hill Climbing Search:

All the successors have h value lower than the current state. But it's possible that all the successors have same h value as current state.This is called as 'Plateau' or 'Flat local maxima'. In this case the search algo may get stuck as it keeps on finding better neighbour and it does not get one. This problem can be solved with following Hill Climbing Search algorithm variants.

a) Sideways Moves:

In this variant, the algorithm keeps on selecting any random successor having h value equal to current state's h value.Such sideway walks are done with the hope that there will be a shoulder to climb to the global maxima. However, it's possible that the algorithm may not find any shoulder and get stuck on sideway moves. To overcome this situation, we put a limit on the sideway walks. Meaning, if the algorithm does not find any shoulder within that limit, it should return as a failure.

b) Random restart without sideway moves:

This variant is used to start the algorithm again if the initial state is not a goal state. When the newly generated state becomes the initial state then the algorithm returns success otherwise a failure.

c) Random restart with sideway moves:

In this variant, algorithm checks if the initial state is a goal state. If that is not the case then the algorithm finds better successors which have h value less than the current state. If the algorithm gets stuck into flat local maxima or plateau then it keeps on performing sideway walks till a provided limit in hope of getting a shoulder. If it gets a shoulder to climb to the global maxima then algorithm returns success. If it does not find the shoulder when performing sideway walks within the limit then algorithm restarts with a new initial state and starts over.

6) Checkpoints covered:

Below displayed rubrics is covered as a part of the project.

Item		Points	Group
Hill climbing search		9 ✔	30
	Run several times, say 100 to 500, and report success and failure rates	9 ✔	
	The average number of steps when it succeeds	4 ✔	
	The average number of steps when it fails	4 ✔	
	The search sequences from four random initial configurations	4 ✔	
Hill-climbing search with sideways move		9 ✔	30
	Run several times, say 100 to 500, and report success and failure rates	9 ✔	
	The average number of steps when it succeeds	4 ✔	
	The average number of steps when it fails	4 ✔	
	The search sequences from four random initial configurations	4 ✔	
Random-restart hill-climbing search		9 ✔	25
	The average number of random restarts required without sideways move	4 ✔	
	The average number of steps required without sideways move	4 ✔	
	The average number of random restarts used with sideways move	4 ✔	
	The average number of steps required with sideways move	4 ✔	
Solving n-Queens problem that take user input for n		5 ✔	
Quality of report		10	
	Total	**100**	-

7) Program Structure:

7.1) Total number of main class files: 5.

7.1.1) **Main.java** : This file is the starting point of execution of the program. It accepts number of queens from user. Then displays user choices to select any one Hill Climbing Search algorithm variant at a time to solve the puzzle.

7.1.2) **Board.java** :Resets the boards and then generates a new board by randomly placing the queens.

Functions:

1. **startNewGameBoard() :** Function to start the initial game board.
2. **heuristicCalculationOfBoard(int[][] board) :** Function to calculate the heuristic of the board sent as parameter. The heuristic here is the number of pairs of queens attacking each other moving row wise then searching for attacks same column, two diagonal from both sides.
3. **resetBoards() :** Function to reset the boards and then generates a new board by randomly placing the queens.
4. **setBoard() :** Function to set the gameboard.

7.1.3) HillClimbSteepestAscent.java: This class solves the N-Queen problem using the steepest ascent hill climbing algorithm.

Functions:

1.**executeHillClimbAlgo**(): Function to find the successors having h value smaller than the current state of board. The successor with smallest h value is selected for expansion. If all such successors have same h value < current board's h value then any random successor is picked up for expansion.This process continues until a board with h=0 is found, returning success. If not board with h-0 is found, failure is returned.

Local Variables:

1. noOfSuccessSteps: Total number of steps required to get goal state of board
2. noOfFailSteps: Total number of steps required to conclude that the solution can not be found.

7.1.4) HillClimbSidewaysMove.java: This class solves the N-Queen problem using the Hill Climbing algorithm with sideways move.

Functions:

1. executeHillClimbAlgo(): Function to find the successors having h value smaller than the current state of board. The successor with smallest h value is selected for expansion. If all the successors have the same h value as the current game board then the algorithm performs sideways walk until the limit is reached. Here, the limit it set to 200. If within this limit algorithm can not find any successors with either h=0 or h < current state's h value, then it returns failure. And success if it eventually gets a board with h=0.

Local Variables:

1. noOfSuccessSteps: Total number of steps required to get goal state of board
2. noOfFailSteps: Total number of steps required to conclude that the solution can not be found.
3.sidewaysMovesCompleted: Number of sideway walks completed by the algorithm.

7.1.5) **HillClimbRandomRestart.java** :This class solves the N-Queen problem using the random restart hill climbing algorithm with and without sideways moves.

Functions:

1. runWithoutSideways(): Function to find the goal state having h=0 without performing any sideways moves. If the initial state's h value is not equal to 0, then the algorithm simply starts over to get the initial state which is same as goal state.

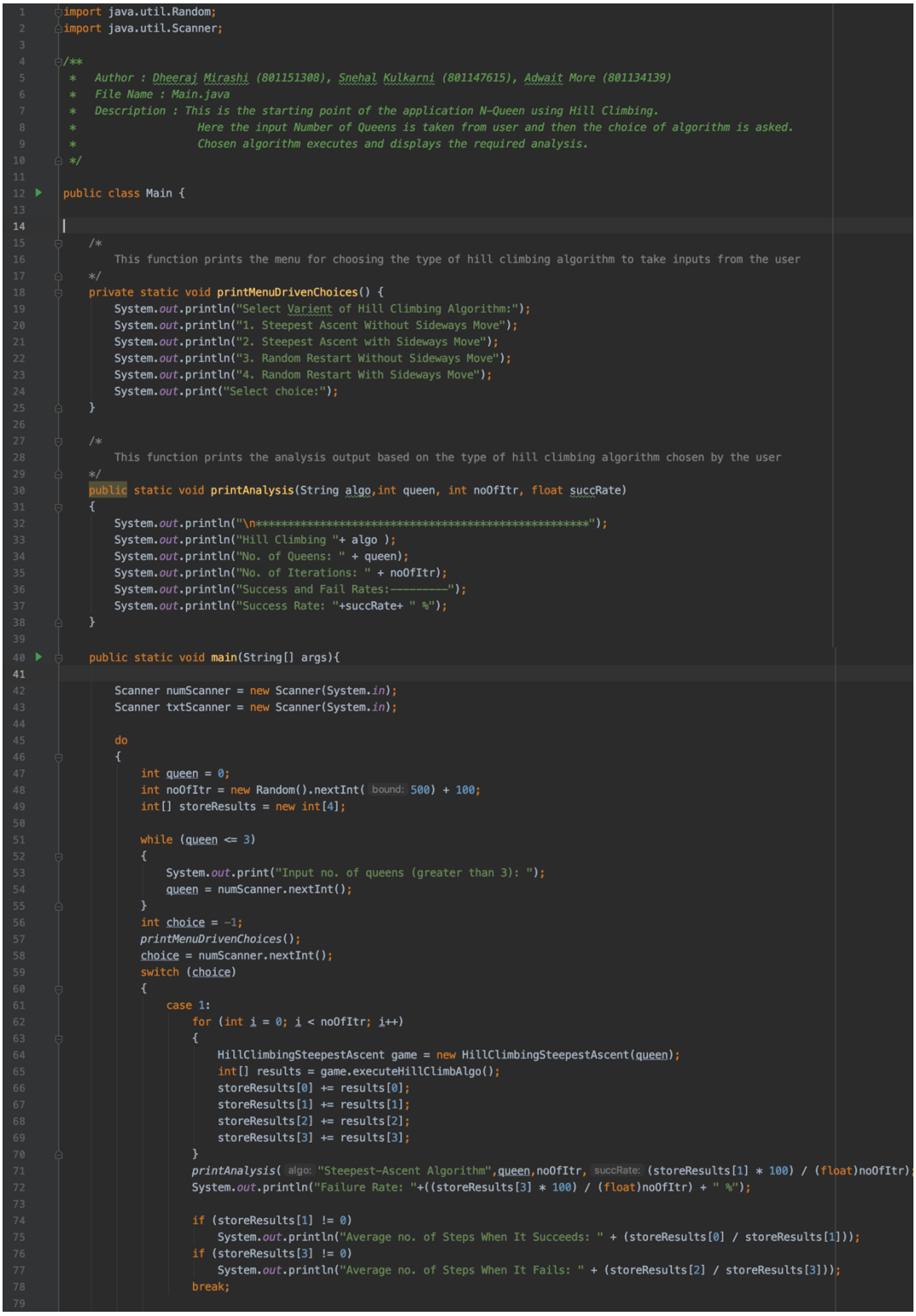
2. runWithSideways(): Function to find the goal state having h=0 by performing sideways moves. If the initial state's h value is not equal to 0, then the algorithm keeps on doing sideways walks until the limit is reached. Here it is set to 100.If the algorithm can not find goal state with h=0 with the limit of 100 sideways walks then it starts over.

Local Variables:

1. noOfSuccessSteps: Total number of steps required to get goal state of board
2.sidewaysMovesCompleted: Number of sideway walks completed by the algorithm.

8. Source Code:

8.1)Main.java:



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

/**
 *   Author : Dheeraj Mirashi (801151308), Snehal Kulkarni (801147615), Adwait More (801134139)
 *   File Name : Main.java
 *   Description : This is the starting point of the application N-Queen using Hill Climbing.
 *                 Here the input Number of Queens is taken from user and then the choice of algorithm is asked.
 *                 Chosen algorithm executes and displays the required analysis.
 */

public class Main {


    /*
        This function prints the menu for choosing the type of hill climbing algorithm to take inputs from the user
    */
    private static void printMenuDrivenChoices() {
        System.out.println("Select Varient of Hill Climbing Algorithm:");
        System.out.println("1. Steepest Ascent Without Sideways Move");
        System.out.println("2. Steepest Ascent with Sideways Move");
        System.out.println("3. Random Restart Without Sideways Move");
        System.out.println("4. Random Restart With Sideways Move");
        System.out.print("Select choice:");
    }

    /*
        This function prints the analysis output based on the type of hill climbing algorithm chosen by the user
    */
    public static void printAnalysis(String algo,int queen, int noOfItr, float succRate)
    {
        System.out.println("\n**************************************************");
        System.out.println("Hill Climbing "+ algo );
        System.out.println("No. of Queens: " + queen);
        System.out.println("No. of Iterations: " + noOfItr);
        System.out.println("Success and Fail Rates:---------");
        System.out.println("Success Rate: "+succRate+ " %");
    }

    public static void main(String[] args){

        Scanner numScanner = new Scanner(System.in);
        Scanner txtScanner = new Scanner(System.in);

        do
        {
            int queen = 0;
            int noOfItr = new Random().nextInt( bound: 500) + 100;
            int[] storeResults = new int[4];

            while (queen <= 3)
            {
                System.out.print("Input no. of queens (greater than 3): ");
                queen = numScanner.nextInt();
            }
            int choice = -1;
            printMenuDrivenChoices();
            choice = numScanner.nextInt();
            switch (choice)
            {
                case 1:
                    for (int i = 0; i < noOfItr; i++)
                    {
                        HillClimbingSteepestAscent game = new HillClimbingSteepestAscent(queen);
                        int[] results = game.executeHillClimbAlgo();
                        storeResults[0] += results[0];
                        storeResults[1] += results[1];
                        storeResults[2] += results[2];
                        storeResults[3] += results[3];
                    }
                    printAnalysis( algo: "Steepest-Ascent Algorithm",queen,noOfItr, succRate: (storeResults[1] * 100) / (float)noOfItr);
                    System.out.println("Failure Rate: "+((storeResults[3] * 100) / (float)noOfItr) + " %");

                    if (storeResults[1] != 0)
                        System.out.println("Average no. of Steps When It Succeeds: " + (storeResults[0] / storeResults[1]));
                    if (storeResults[3] != 0)
                        System.out.println("Average no. of Steps When It Fails: " + (storeResults[2] / storeResults[3]));
                    break;

```

```
                    case 2:
                        for (int i = 0; i < noOfItr; i++)
                        {
                            HillClimbingSidewaysMove game = new HillClimbingSidewaysMove(queen);
                            int[] results = game.executeHillClimbAlgo();
                            storeResults[0] += results[0];
                            storeResults[1] += results[1];
                            storeResults[2] += results[2];
                            storeResults[3] += results[3];
                        }

                        printAnalysis( algo: "Sideways Moves Algorithm",queen,noOfItr, succRate: (storeResults[1] * 100) / (float)noOfItr);
                        System.out.println("Failure Rate: "+((storeResults[3] * 100) / (float)noOfItr) + " %");

                        if (storeResults[1] != 0)
                            System.out.println("Average no. of Steps When It Succeeds: " + (storeResults[0] / storeResults[1]));
                        if (storeResults[3] != 0)
                            System.out.println("Average no. of Steps When It Fails: " + (storeResults[2] / storeResults[3]));
                        break;

                    case 3:
                        for (int i = 0; i < noOfItr; i++)
                        {
                            HillClimbingRandomRestart game = new HillClimbingRandomRestart(queen);
                            int[] results = game.runWithoutSideways();
                            storeResults[0] += results[0];
                            storeResults[1] += results[1];
                            storeResults[2] += results[2];
                        }

                        printAnalysis( algo: "Random Restart Algorithm",queen, noOfItr, succRate: (storeResults[1] * 100) / (float)noOfItr );
                        System.out.println("Average no. of Steps When It Succeeds: " + (storeResults[0] / noOfItr));
                        System.out.println("Average no. of Restarts When It Succeeds: " + (storeResults[2] / noOfItr));
                        break;

                    case 4:
                        for (int i = 0; i < noOfItr; i++)
                        {
                            HillClimbingRandomRestart game = new HillClimbingRandomRestart(queen);
                            int[] results = game.runWithSideways();
                            storeResults[0] += results[0];
                            storeResults[1] += results[1];
                            storeResults[2] += results[2];
                        }

                printAnalysis( algo: "Random Restart Algorithm with sideways move",queen,noOfItr,((storeResults[1] * 100) / (float)noOfItr));
                System.out.println("Average no. of steps when it succeeds: " + (storeResults[0] / noOfItr));

                if ((storeResults[2] / (float)noOfItr) > 0 && (storeResults[2] / (float)noOfItr) < 1)
                    System.out.println("Average no. of Restarts when it succeeds: ~ 1");
                else
                    System.out.println("Average no. of Restarts when it succeeds: " + (storeResults[2] / (float)noOfItr));
                break;
            }
            System.out.println("Continue algorithm?  Y or N");
        } while (!txtScanner.nextLine().toUpperCase().equals("N"));
    }


}
```

8.2)Board.java:

```java
/**
 *   Author : Dheeraj Mirashi (801151308), Snehal Kulkarni (801147615), Adwait More (801134139)
 *   File Name : Board.java
 *   Description : This is the base class consisting of all the Board related operations and variables used accross the algorithms
 */

import java.util.Random;

public class Board {

    Random rand = new Random();
    private int queens;
    private int[][] boardOne;
    private int[][] boardTwo;
    private int[][] boardThree;
    public int[][] board;
    public int numberOfResets = -1;
    public int regularMoves = 0;
    public int sidewaysMoves = 0;

    public Board(int numberOfQueens) {
        this.queens = numberOfQueens;
        boardOne = new int[numberOfQueens][numberOfQueens];
        boardTwo = new int[numberOfQueens][numberOfQueens];
        boardThree = new int[numberOfQueens][numberOfQueens];
        this.resetBoards();
        this.setBoard();
    }

    public int getQueens() { return queens; }

    public int[][] getBoardOne() { return boardOne; }

    public int[][] getBoardTwo() { return boardTwo; }

    public int[][] getBoardThree() { return boardThree; }

    public int[][] getBoard() { return board; }

    public int getNumberOfResets() { return numberOfResets; }

    public int getRegularMoves() { return regularMoves; }

    public int getSidewaysMoves() { return sidewaysMoves; }
    public void resetRegularMoves() { this.regularMoves = 0; }

    public void sidewaysMovesReset() { this.sidewaysMoves = 0; }

    /*
       Description : Resets the boards and then generates a new board by randomly placing the queen
    */
    public void resetBoards()
    {
        boardOne = resetOneBoard();
        boardTwo = resetOneBoard();
        boardThree = resetOneBoard();
        this.numberOfResets += 1;
    }

    public void setBoard()
    {
        int n = rand.nextInt( bound: 2);
        if (n == 0)
            this.board = this.createCopy(boardOne);
        else if (n == 1)
            this.board = this.createCopy(boardTwo);
        else
            this.board = this.createCopy(boardThree);
    }

    public int[][] createCopy(int[][] oldBoard)
    {
        int[][] newBoard = new int[this.getQueens()][this.getQueens()];

        for (int i = 0; i < oldBoard.length; i++)
        {
            for (int j = 0; j < oldBoard.length; j++)
            {
                newBoard[i][j] = oldBoard[i][j];
            }
        }
        return newBoard;
    }

```

```java
/*
       Description : This function is starts the game boards
   */
public void startNewGameBoard()
{
    System.out.println("\n\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
    System.out.println("Initial Board");
    System.out.println("State's Heuristic value: " + this.heuristicCalculationOfBoard(this.board));

    for (int x = 0; x < this.board.length; x++)
    {
        for (int y = 0; y < this.board.length; y++)
        {
            System.out.print(this.board[x][y] + " ");
        }
        System.out.println();
    }
}


/*
   Description : This function is prints the game board
*/

public void displayBoard()
{
    System.out.println("State's Heuristic value: " + this.heuristicCalculationOfBoard(this.board));

    for (int x = 0; x < this.board.length; x++)
    {
        for (int y = 0; y < this.board.length; y++)
        {
            System.out.print(this.board[x][y] + " ");
        }
        System.out.println();
    }
}


/*
   Description : This function checks whether the boards are equal or not
*/
public boolean equalsBoards(int[][] boardOne, int[][] boardTwo)
{
    Boolean areEqual = true;

    for (int i = 0; i < this.getQueens(); i++)
    {
        for (int j = 0; j < this.getQueens(); j++)
        {
            if (boardOne[i][j] != boardTwo[i][j])
            areEqual = false;
        }
    }
    return areEqual;
}

/*
  Description : Resets board by randomly placing the queens and then returns it
*/
public int[][] resetOneBoard(){
    int[][] board1 = new int[queens][queens];
    for (int i = 0; i < this.queens; i++)
    {
        int queenPositionY = rand.nextInt( bound: this.queens - 1);

        for (int j = 0; j < this.queens; j++)
        {
            if (j == queenPositionY){
                board1[j][i] = 1; //means queen is placed;
            }
            else{
                board1[j][i] = 0; //means queen is not placed;
            }
        }
    }
    return board1;
}
```

```java
    /*
      Parameters : queenBoard
      Description : This function is calculates the heuristic of the board sent as parameter.
                    The heuristic here is the number of pairs of queens attacking each other
      Returns : Heuristic value (Integer)
    */

    public int heuristicCalculationOfBoard(int[][] queenBoard)
    {
        int[][] board = queenBoard;
        int heuristic = 0;

        for (int i = 0; i < this.queens; i++)
        {
            int noQueens = 0;
            for (int j = 0; j < this.queens; j++)
            {
                if (board[i][j] == 1)
                    noQueens += 1;
            }

            if (noQueens > 1)
            {
                for (int queens = noQueens; queens > 1; --queens)
                {
                    heuristic += queens - 1;
                }
            }
        }


        for (int i = 0; i < this.queens; i++)
        {
            int noQueens = 0;
            for (int j = 0; j < this.queens; j++)
            {
                if (board[j][i] == 1)
                    noQueens += 1;
            }

            if (noQueens > 1)
            {
                for (int queens = noQueens; queens > 1; --queens)
                {
                    heuristic += queens - 1;
                }
            }
        }


        int numOfQueens = 0;
        int i1 = 0;
        while (i1 < (this.queens - 1))
        {
            numOfQueens = 0;
            int xValue = 0;
            int yValue = i1;

            while (xValue <= i1)
            {
                if (board[xValue][yValue] == 1)
                    numOfQueens += 1;

                yValue--;
                xValue++;
            }
            if (numOfQueens > 1)
            {
                for (int queens = numOfQueens; queens > 1; --queens)
                {
                    heuristic += queens - 1;
                }
            }
            i1++;
        }


        numOfQueens = 0;
        int diagonal = this.queens - 1;
        for (int i = 0; i < this.queens; i++)
        {
            if (board[i][diagonal] == 1)
                numOfQueens++;
            diagonal--;
        }
```

```
273
274         if (numOfQueens > 1)
275         {
276             for (int queens = numOfQueens; queens > 1; --queens)
277             {
278                 heuristic += queens - 1;
279             }
280         }
281
282
283         numOfQueens = 0;
284         i1 = 1;
285         while (i1 < (this.queens - 1))
286         {
287             numOfQueens = 0;
288             int xValue = i1;
289             int yValue = this.queens - 1;
290
291             while (xValue < this.queens)
292             {
293                 if (board[xValue][yValue] == 1)
294                     numOfQueens += 1;
295
296                 yValue--;
297                 xValue++;
298             }
299             if (numOfQueens > 1)
300             {
301                 for (int queens = numOfQueens; queens > 1; --queens)
302                 {
303                     heuristic += queens - 1;
304                 }
305             }
306
307             i1++;
308         }
309
310
311         numOfQueens = 0;
312         i1 = this.queens - 2;
313         while (i1 > 0)
314         {
315             numOfQueens = 0;
316             int xValue = 0;
317             int yValue = i1;
318
319             while (yValue < this.queens)
320             {
321                 if (board[xValue][yValue] == 1)
322                     numOfQueens += 1;
323
324                 yValue++;
325                 xValue++;
326             }
327
328             if (numOfQueens > 1)
329             {
330                 for (int queens = numOfQueens; queens > 1; --queens)
331                 {
332                     heuristic += queens - 1;
333                 }
334             }
335
336             i1--;
337         }
338
339
340         numOfQueens = 0;
341         diagonal = 0;
342         for (int i = 0; i < this.queens; i++)
343         {
344             if (board[i][diagonal] == 1)
345                 numOfQueens++;
346             diagonal++;
347         }
348
349         if (numOfQueens > 1)
350         {
351             for (int queens = numOfQueens; queens > 1; --queens)
352             {
353                 heuristic += queens - 1;
354             }
355         }
```

```
356
357
358             numOfQueens = 0;
359             i1 = 1;
360             while (i1 < this.queens)
361             {
362                 numOfQueens = 0;
363                 int xValue = i1;
364                 int yValue = 0;
365
366                 while (xValue < this.queens)
367                 {
368                     if (board[xValue][yValue] == 1)
369                         numOfQueens += 1;
370
371                     yValue++;
372                     xValue++;
373                 }
374                 if (numOfQueens > 1)
375                 {
376                     for (int queens = numOfQueens; queens > 1; --queens)
377                     {
378                         heuristic += queens - 1;
379                     }
380                 }
381
382                 i1++;
383             }
384
385             return heuristic;
386         }
387     }
```

8.3)HillClimbSteepestAscent.java

```
1   import java.util.ArrayList;
2   import java.util.Random;
3
4   /**
5    *   Author : Dheeraj Mirashi (801151308), Snehal Kulkarni (801147615), Adwait More (801134139)
6    *   File Name : HillClimbingSteepestAscent.java
7    *   Description : This class solves the N-Queen problem using the steepest ascent hill climbing algorithm
8    */
9
10  public class HillClimbingSteepestAscent extends Board {
11
12      private int noOfSuccessSteps = 0;
13      private int noSuccessIterations = 0;
14      private int noOfFailSteps = 0;
15      private int noFailIterations = 0;
16      private boolean changesInBoard = true;
17
18      public HillClimbingSteepestAscent(int numberOfQueens) { super(numberOfQueens); }
21
22      /*
23         Description : This function is called untill the heiuristic value is zero and neighbours with a lower heuristic value is found
24                       This is failed if heuristic value equal to zero is not found
25         Returns : Success / Failure Analysis
26       */
27
28      public int[] executeHillClimbAlgo()
29      {
30          this.startNewGameBoard();
31          this.changesInBoard = true;
32          int currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
33          ArrayList<int[][]> possibleMoves = new ArrayList<int[][]>();
34          this.resetRegularMoves();
35
36          while (this.heuristicCalculationOfBoard(this.board) != 0 && (this.changesInBoard == true))
37          {
38              int columnNumber = 0;
39              int[][] possibleMove = new int[this.getQueens()][this.getQueens()];
40              for (int i = 0; i < this.getQueens(); i++)
41              {
42                  for (int j = 0; j < this.getQueens(); j++)
43                  {
44                      possibleMove[i][j] = this.board[i][j];
45                  }
46              }
47
```

```
            currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
            this.changesInBoard = false;

            while (columnNumber < this.getQueens())
            {
                int currentColumnQueenPosition = -1;

                for (int i = 0; i < this.getQueens(); i++)
                {
                    if (possibleMove[i][columnNumber] == 1)
                    currentColumnQueenPosition = i;
                    possibleMove[i][columnNumber] = 0;
                }

                for (int i = 0; i < this.getQueens(); i++)
                {
                    possibleMove[i][columnNumber] = 1;

                    int[][] newMove = new int[this.getQueens()][this.getQueens()];
                    for (int k = 0; k < this.getQueens(); k++)
                    {
                        for (int j = 0; j < this.getQueens(); j++)
                        {
                            newMove[k][j] = possibleMove[k][j];
                        }
                    }
                    if (this.heuristicCalculationOfBoard(this.board) > this.heuristicCalculationOfBoard(newMove))
                        possibleMoves.add(newMove);
                    possibleMove[i][columnNumber] = 0;
                }

                possibleMove[currentColumnQueenPosition][columnNumber] = 1;

                columnNumber += 1;
            }

            Random rand = new Random();

            if (possibleMoves.size() != 0)
            {
                int pick = rand.nextInt(possibleMoves.size());

                int minHeuristic = currentStateHeuristic;

                if (minHeuristic > this.heuristicCalculationOfBoard(possibleMoves.get(pick)))
                {
                    minHeuristic = this.heuristicCalculationOfBoard(possibleMoves.get(pick));
                    this.board = this.createCopy(possibleMoves.get(pick));
                    this.changesInBoard = true;

                    this.regularMoves += 1;
                    System.out.println("\nStep No." + regularMoves);
                    this.displayBoard();
                    possibleMoves.clear();
                }
            }
            else
            {
                if (possibleMoves.size() == 0)
                {
                    this.changesInBoard = false;
                    possibleMoves.clear();
                }
            }
        }

        if (this.heuristicCalculationOfBoard(this.board) == 0)
        {
            System.out.println("\nSuccess!");
            this.noOfSuccessSteps += this.regularMoves;
            this.noSuccessIterations += 1;
             System.out.println("Total Steps Taken: " + this.regularMoves);
        }
        else
        {
             System.out.println("\nFailure!");
            this.displayBoard();
            this.noOfFailSteps += this.regularMoves;
            this.noFailIterations += 1;
             System.out.println("Total Steps Taken: " + this.regularMoves);
        }

        return new int[] {noOfSuccessSteps, noSuccessIterations, noOfFailSteps,
                noFailIterations};
    }
}
```

8.4)HillClimbSidewaysMove.java

```java
import java.util.ArrayList;
import java.util.Random;

/**
 *   Author : Dheeraj Mirashi (801151308), Snehal Kulkarni (801147615), Adwait More (801134139)
 *   File Name : HillClimbingSidewaysMove.java
 *   Description : This class solves the N-Queen problem using the Hill Climbing algorithm with sideways move.
 */

public class HillClimbingSidewaysMove extends Board {
    private int noOfSuccessSteps = 0;
    private int noSuccessIterations = 0;
    private int noOfFailSteps = 0;
    private int noFailIterations = 0;
    private boolean changesInBoard = true;
    private int sidewaysMovesConsecutive = 0;
    private final int LIMIT_OF_SIDEWAYS_MOVES = 200;

    public HillClimbingSidewaysMove(int numberOfQueens) { super(numberOfQueens); }

    /*
       Description : This function is called until the heuristic value is zero and neighbours with a lower heuristic value is found
       Returns : Success / Failure Analysis
     */

    public int[] executeHillClimbAlgo()
    {
        this.startNewGameBoard();
        this.changesInBoard = true;
        int currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
        ArrayList<int[][]> possibleMoves = new ArrayList<int[][]>();
        this.resetRegularMoves();

        while (this.heuristicCalculationOfBoard(this.board) != 0 && (this.changesInBoard == true))
        {
            Boolean moveMade = false;
            int columnNumber = 0;
            int[][] possibleMove = new int[this.getQueens()][this.getQueens()];
            for (int i = 0; i < this.getQueens(); i++)
            {
                for (int j = 0; j < this.getQueens(); j++)
                {
                    possibleMove[i][j] = this.board[i][j];
                }
            }
            currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
            this.changesInBoard = false;

            while (columnNumber < this.getQueens())
            {
                int currentColumnQueenPosition = -1;

                for (int i = 0; i < this.getQueens(); i++)
                {
                    if (possibleMove[i][columnNumber] == 1)
                    currentColumnQueenPosition = i;
                    possibleMove[i][columnNumber] = 0;
                }

                for (int i = 0; i < this.getQueens(); i++)
                {
                    possibleMove[i][columnNumber] = 1;

                    int[][] newMove = new int[this.getQueens()][this.getQueens()];
                    for (int k = 0; k < this.getQueens(); k++)
                    {
                        for (int j = 0; j < this.getQueens(); j++)
                        {
                            newMove[k][j] = possibleMove[k][j];
                        }
                    }
                    if (this.heuristicCalculationOfBoard(this.board) >= this.heuristicCalculationOfBoard(newMove) && this.equalsBoards(this.board, newMove) == false)
                        possibleMoves.add(newMove);
                    possibleMove[i][columnNumber] = 0;
                }

                possibleMove[currentColumnQueenPosition][columnNumber] = 1;

                columnNumber += 1;
            }

            Random rand = new Random();
            int minHeuristic = currentStateHeuristic;

            if (possibleMoves.size() != 0)
            {
                int pick = rand.nextInt(possibleMoves.size());

```

```
                if (minHeuristic > this.heuristicCalculationOfBoard(possibleMoves.get(pick)))
                {
                    minHeuristic = this.heuristicCalculationOfBoard(possibleMoves.get(pick));
                    this.board = this.createCopy(possibleMoves.get(pick));
                    this.changesInBoard = true;
                    this.regularMoves += 1;
                    this.sidewaysMovesConsecutive = 0;
                    System.out.println("\nStep No." + regularMoves);
                    this.displayBoard();
                    possibleMoves.clear();
                }
                else if (minHeuristic == this.heuristicCalculationOfBoard(possibleMoves.get(pick)) &&
                        this.sidewaysMovesConsecutive < 200)    //restriction to number of sideways move
                {
                    minHeuristic = this.heuristicCalculationOfBoard(possibleMoves.get(pick));
                    this.board = this.createCopy(possibleMoves.get(pick));
                    this.changesInBoard = true;
                    this.sidewaysMovesConsecutive += 1;
                    this.regularMoves += 1;
                    System.out.println("\nStep No." + regularMoves);
                    this.displayBoard();
                    System.out.println("Sideways Move!");
                    possibleMoves.clear();
                }
            }
            else
            {
                this.changesInBoard = false;
            }
        }

        if (this.heuristicCalculationOfBoard(this.board) == 0)
        {
            System.out.println("\nSuccess!");
            this.noOfSuccessSteps += this.regularMoves;
            this.noSuccessIterations += 1;
            System.out.println("Total Steps Taken: " + this.regularMoves);
        }
        else
        {
            System.out.println("\nFailure!");
            this.displayBoard();
            this.noOfFailSteps += this.regularMoves;
            this.noFailIterations += 1;
            System.out.println("Total Steps Taken: " + this.regularMoves);
        }

        return new int[] {noOfSuccessSteps, noSuccessIterations, noOfFailSteps,
                noFailIterations};
    }
}
```

8.5)HillClimbRandomRestart.java

```
import java.util.ArrayList;
import java.util.Random;

/**
 *  Author : Dheeraj Mirashi (801151308), Snehal Kulkarni (801147615), Adwait More (801134139)
 *  File Name : HillClimbingRandomRestart.java
 *  Description : This class solves the N-Queen problem using the random restart hill climbing algorithm with and without sideways
 */


public class HillClimbingRandomRestart extends Board {
    private int noOfSuccessSteps = 0;
    private int noSuccessIterations = 0;
    private boolean boardChanged = true;
    private int sidewaysMovesCompleted = 0;

    public HillClimbingRandomRestart(int numberOfQueens) { super(numberOfQueens); }

    /*
       Description : Resetboard from Board.java is called, and game board is restarted with side ways move
                     whenever board heuristic is stuck at h > 0
       Returns : Success / Failure Analysis
     */

    public int[] runWithSideways()
    {
        this.startNewGameBoard();
        boolean boardChanged = true;
        int currStateHeuristic = this.heuristicCalculationOfBoard(this.board);
        ArrayList<int[][]> listOfPossibleMoves = new ArrayList<int[][]>();
        this.resetRegularMoves();

        while (this.heuristicCalculationOfBoard(this.board) != 0 && boardChanged == true)
        {
            int columnNumber = 0;
            int[][] possibleMove = new int[this.getQueens()][this.getQueens()];
            for (int i = 0; i < this.getQueens(); i++)
            {
                for (int j = 0; j < this.getQueens(); j++)
                {
                    possibleMove[i][j] = this.board[i][j];
                }
            }

```

```
        currStateHeuristic = this.heuristicCalculationOfBoard(this.board);
        this.boardChanged = false;

        while (columnNumber < this.getQueens())
        {
            int currentQueenPos = -1;

            for (int i = 0; i < this.getQueens(); i++)
            {
                if (possibleMove[i][columnNumber] == 1)
                    currentQueenPos = i;
                possibleMove[i][columnNumber] = 0;
            }

            for (int i = 0; i < this.getQueens(); i++)
            {
                possibleMove[i][columnNumber] = 1;

                int[][] newMoveOfQueen = new int[this.getQueens()][this.getQueens()];
                for (int k = 0; k < this.getQueens(); k++)
                {
                    for (int j = 0; j < this.getQueens(); j++)
                    {
                        newMoveOfQueen[k][j] = possibleMove[k][j];
                    }
                }
                if (this.heuristicCalculationOfBoard(this.board) >= this.heuristicCalculationOfBoard(newMoveOfQueen) && this.equalsBoards(this.board, newMoveOfQueen) == false)
                    listOfPossibleMoves.add(newMoveOfQueen);
                possibleMove[i][columnNumber] = 0;
            }

            possibleMove[currentQueenPos][columnNumber] = 1;

            columnNumber += 1;
        }

        Random rand = new Random();
        int minHeuristic = currStateHeuristic;

        if (listOfPossibleMoves.size() != 0)
        {
            int pick = rand.nextInt(listOfPossibleMoves.size());
            if (minHeuristic > this.heuristicCalculationOfBoard(listOfPossibleMoves.get(pick)))
            {
                minHeuristic = this.heuristicCalculationOfBoard(listOfPossibleMoves.get(pick));
                this.board = this.createCopy(listOfPossibleMoves.get(pick));
                this.boardChanged = true;
                this.sidewaysMovesCompleted = 0;
                this.regularMoves += 1;
                System.out.println("\nStep No." + regularMoves);
                this.displayBoard();
                listOfPossibleMoves.clear();
            }
            else if (minHeuristic == this.heuristicCalculationOfBoard(listOfPossibleMoves.get(pick)) &&
                    this.sidewaysMovesCompleted < 100) //restricting the sideways moves to 100
            {
                minHeuristic = this.heuristicCalculationOfBoard(listOfPossibleMoves.get(pick));
                this.board = this.createCopy(listOfPossibleMoves.get(pick));
                this.boardChanged = true;
                this.sidewaysMovesCompleted += 1;
                this.regularMoves += 1;
                System.out.println("\nStep No." + regularMoves);
                this.displayBoard();
                System.out.println("Sideways Move!");
                listOfPossibleMoves.clear();
            }
            else
            {
                this.resetBoards();
                this.setBoard();
                this.boardChanged = true;
                listOfPossibleMoves.clear();
                System.out.println("Random Restart!");
                this.startNewGameBoard();
            }
        }
        else
        {
            this.resetBoards();
            this.setBoard();
            this.boardChanged = true;
            listOfPossibleMoves.clear();
            System.out.println("Random Restart!");
            this.startNewGameBoard();
        }
    }

```

```
        updateSuccessStatusBasedOnHeuristic();

        return new int[] {noOfSuccessSteps, noSuccessIterations, this.getNumberOfResets() };
    }

    /*
       Description : Resetboard from Board.java is called, and game board is restarted whenever board heuristic is stuck at h > 0
       Returns : Success / Failure Analysis
     */

    public int[] runWithoutSideways()
    {
        this.startNewGameBoard();
        boolean boardChanged = true;
        int currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
        ArrayList<int[][]> possibleMoves = new ArrayList<int[][]>();
        this.resetRegularMoves();

        while (this.heuristicCalculationOfBoard(this.board) != 0 && boardChanged == true)
        {
            int columnNumber = 0;
            int[][] possibleMove = new int[this.getQueens()][this.getQueens()];
            for (int i = 0; i < this.getQueens(); i++)
            {
                for (int j = 0; j < this.getQueens(); j++)
                {
                    possibleMove[i][j] = this.board[i][j];
                }
            }

            currentStateHeuristic = this.heuristicCalculationOfBoard(this.board);
            this.boardChanged = false;

            while (columnNumber < this.getQueens())
            {
                int currentColumnQueenPosition = -1;

                for (int i = 0; i < this.getQueens(); i++)
                {
                    if (possibleMove[i][columnNumber] == 1)
                    currentColumnQueenPosition = i;
                    possibleMove[i][columnNumber] = 0;
                }

                for (int i = 0; i < this.getQueens(); i++)
                {
                    possibleMove[i][columnNumber] = 1;

                    int[][] newMove = new int[this.getQueens()][this.getQueens()];
                    for (int k = 0; k < this.getQueens(); k++)
                    {
                        for (int j = 0; j < this.getQueens(); j++)
                        {
                            newMove[k][j] = possibleMove[k][j];
                        }
                    }
                    if (this.heuristicCalculationOfBoard(this.board) > this.heuristicCalculationOfBoard(newMove))
                        possibleMoves.add(newMove);
                    possibleMove[i][columnNumber] = 0;
                }

                possibleMove[currentColumnQueenPosition][columnNumber] = 1;

                columnNumber += 1;
            }

            Random rand = new Random();

            if (possibleMoves.size() != 0)
            {
                int pick = rand.nextInt(possibleMoves.size());

                int minHeuristic = currentStateHeuristic;

                if (minHeuristic > this.heuristicCalculationOfBoard(possibleMoves.get(pick)))
                {
                    minHeuristic = this.heuristicCalculationOfBoard(possibleMoves.get(pick));
                    this.board = this.createCopy(possibleMoves.get(pick));
                    this.boardChanged = true;

                    this.regularMoves += 1;
                    System.out.println("\nStep No." + regularMoves);
                    this.displayBoard();
                    possibleMoves.clear();
                }
            }
```

```java
            else
            {
                this.resetBoards();
                this.setBoard();
                this.boardChanged = true;
                possibleMoves.clear();
                System.out.println("Random Restart!");
                this.startNewGameBoard();
            }
        }

        updateSuccessStatusBasedOnHeuristic();

        return new int[] {noOfSuccessSteps, noSuccessIterations, this.getNumberOfResets() };
    }

    /*
        Description : After calculation of the heuristic, this function displays the success status for both of the above methods
     */
    private void updateSuccessStatusBasedOnHeuristic(){
        if (this.heuristicCalculationOfBoard(this.board) == 0)
        {
            System.out.println("\nSuccess!");
            this.noOfSuccessSteps += this.regularMoves;
            this.noSuccessIterations += 1;
            System.out.println("Total Steps Taken: " + this.regularMoves);
        }
    }
}
```

9. Inputs and Outputs:

1. Hill Climbing Steepest-Ascent Algorithm

Input no. of queens (greater than 3): 6
Select Varient of Hill Climbing Algorithm:
1. Steepest Ascent Without Sideways Move
2. Steepest Ascent with Sideways Move
3. Random Restart Without Sideways Move
4. Random Restart With Sideways Move
Select choice:1

>>>
Initial Board
State's Heuristic value: 6
0 0 0 0 0 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 0 0 0 0

Step No.1
State's Heuristic value: 4
1 0 0 0 0 0
0 0 1 1 0 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

Step No.2
State's Heuristic value: 3
1 0 0 1 0 0

0 0 1 0 0 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

Step No.3
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0

Failure!
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
Total Steps Taken: 3

>>
Initial Board
State's Heuristic value: 6
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 1 0 1 1
1 0 0 1 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 5
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 1
1 0 0 1 0 0
0 0 0 0 1 0

Step No.2
State's Heuristic value: 4
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 1
1 0 0 1 0 0
0 0 0 0 0 0

Step No.3
State's Heuristic value: 2

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

Step No.4
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 0 0 0

Failure!
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 0 0 0
Total Steps Taken: 4

>>>
Initial Board
State's Heuristic value: 6
0 1 0 0 1 1
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 1 0 0
1 0 0 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 5
0 1 0 0 1 1
0 0 0 0 0 0
1 0 0 0 0 0
0 0 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Step No.2
State's Heuristic value: 4
0 1 0 0 1 1
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Step No.3

State's Heuristic value: 3
0 1 0 0 1 1
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 0 1 0 0 0

Step No.4
State's Heuristic value: 2
0 1 0 0 0 1
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 0 1 0 1 0

Failure!
State's Heuristic value: 2
0 1 0 0 0 1
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 0 1 0 1 0
Total Steps Taken: 4

>>>
Initial Board
State's Heuristic value: 6
0 0 0 0 0 0
0 1 0 1 0 1
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 5
0 0 0 0 0 0
0 1 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 1
0 0 0 0 0 0

Step No.2
State's Heuristic value: 4
0 0 0 0 0 0
0 1 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1

Step No.3
State's Heuristic value: 3
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 0
1 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1

Step No.4
State's Heuristic value: 2
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0
1 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0

Step No.5
State's Heuristic value: 1
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0

Failure!
State's Heuristic value: 1
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Total Steps Taken: 5

>>

Hill Climbing Steepest-Ascent Algorithm
No. of Queens: 6
No. of Iterations: 352
Success and Fail Rates:---------
Success Rate: 9.375 %
Failure Rate: 90.625 %
Average no. of Steps When It Succeeds: 4
Average no. of Steps When It Fails: 3

2. Hill Climbing Sideways Moves Algorithm

Initial Board
State's Heuristic value: 6
0 1 0 0 1 0
0 0 0 1 0 0
0 0 0 0 0 0

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

Step No.1
State's Heuristic value: 6
0 1 0 1 1 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 0 0 0
Sideways Move!

Step No.2
State's Heuristic value: 5
0 1 0 1 1 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 1
1 0 0 0 0 0
0 0 0 0 0 0

Step No.3
State's Heuristic value: 3
0 1 0 1 0 0
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 1
1 0 0 0 0 0
0 0 0 0 0 0

Step No.4
State's Heuristic value: 3
0 1 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 1
1 0 0 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.5
State's Heuristic value: 3
0 1 0 1 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.6
State's Heuristic value: 3
0 1 0 1 0 0
0 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
Sideways Move!

Step No.7
State's Heuristic value: 3
0 0 0 1 0 0
0 1 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
Sideways Move!

Step No.8
State's Heuristic value: 2
0 0 0 1 0 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0

Step No.9
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0

Step No.10
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.11
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.12
State's Heuristic value: 1

1 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.13
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.14
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.15
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.16
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.17
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0

Sideways Move!

Step No.18
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.19
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.20
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.21
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.22
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.23
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0

0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.24
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.25
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.26
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.27
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.28
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.29
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.30
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.31
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.32
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.33
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.34
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1

0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.35
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.36
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 1 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.37
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.38
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.39
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.40
State's Heuristic value: 1

0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.41
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 1 0
Sideways Move!

Step No.42
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 1 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.43
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.44
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.45
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0

Sideways Move!

Step No.46
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.47
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.48
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.49
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.50
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.51
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0

0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.52
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.53
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.54
State's Heuristic value: 1
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 1 1 0 0 0
Sideways Move!

Step No.55
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.56
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.57
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.58
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.59
State's Heuristic value: 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.60
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.61
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.62
State's Heuristic value: 1
0 0 0 0 0 1
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0

0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.63
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.64
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.65
State's Heuristic value: 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0

Success!
Total Steps Taken: 65

>>
Initial Board
State's Heuristic value: 7
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 1 1 0
0 0 1 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 4
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 1 1 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0

Step No.2
State's Heuristic value: 3
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 1 0
0 0 1 0 0 0

Step No.3
State's Heuristic value: 3
0 0 0 0 0 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 1 0
0 0 1 0 0 0
Sideways Move!

Step No.4
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0

Step No.5
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.6
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.7
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 1 0 0 0
0 0 0 0 1 0

Sideways Move!

Step No.8
State's Heuristic value: 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.9
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.10
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.11
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.12
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 1
Sideways Move!

Step No.13
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0

0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.14
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.15
State's Heuristic value: 1
0 0 0 0 0 1
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.16
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.17
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.18
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.19
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.20
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.21
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.22
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.23
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.24
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0

0 0 1 0 0 0
0 0 0 1 0 0
Sideways Move!

Step No.25
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.26
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.27
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.28
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.29
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.30
State's Heuristic value: 1

0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.31
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.32
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.33
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.34
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.35
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0

Sideways Move!

Step No.36
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.37
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.38
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.39
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.40
State's Heuristic value: 1
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 1 1 0 0 0
Sideways Move!

Step No.41
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0

0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.42
State's Heuristic value: 1
0 0 0 0 1 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.43
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.44
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.45
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.46
State's Heuristic value: 1
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
1 0 0 0 0 0
Sideways Move!

Step No.47
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.48
State's Heuristic value: 1
0 0 0 0 1 0
0 1 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.49
State's Heuristic value: 1
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
1 1 0 0 0 0
Sideways Move!

Step No.50
State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Success!
Total Steps Taken: 50

>>
Initial Board
State's Heuristic value: 7
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 1 0
0 0 0 0 0 1
1 0 1 1 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 7
0 0 0 0 0 0

0 0 0 0 0 0
0 1 0 0 1 0
0 0 1 0 0 1
1 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.2
State's Heuristic value: 7
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 1 0
0 0 1 1 0 1
1 0 0 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.3
State's Heuristic value: 6
0 0 0 0 0 0
0 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 0 0
0 0 0 0 0 0

Step No.4
State's Heuristic value: 6
0 0 0 0 0 0
0 0 1 0 0 1
0 1 0 0 1 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.5
State's Heuristic value: 6
0 0 0 0 0 0
0 0 1 0 0 1
0 1 0 0 1 0
0 0 0 0 0 0
1 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.6
State's Heuristic value: 6
0 0 0 0 1 0
0 0 1 0 0 1
0 1 0 0 0 0
0 0 0 0 0 0
1 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.7
State's Heuristic value: 4
0 0 0 0 1 0
0 0 1 0 0 1
0 1 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0

Step No.8
State's Heuristic value: 4
0 0 0 0 0 0
0 0 1 0 1 1
0 1 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
Sideways Move!

Step No.9
State's Heuristic value: 4
1 0 0 0 0 0
0 0 1 0 1 1
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
Sideways Move!

Step No.10
State's Heuristic value: 4
1 0 0 0 0 0
0 0 1 0 1 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
Sideways Move!

Step No.11
State's Heuristic value: 3
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0

Step No.12
State's Heuristic value: 3
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
0 0 0 0 0 0

Sideways Move!

Step No.13
State's Heuristic value: 2
1 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 1 0 1
0 0 1 0 0 0

Step No.14
State's Heuristic value: 2
1 0 0 0 0 0
0 0 0 0 1 0
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.15
State's Heuristic value: 2
1 0 0 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.16
State's Heuristic value: 2
1 0 0 1 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 1 0 0 0
Sideways Move!

Step No.17
State's Heuristic value: 2
1 0 0 0 0 0
0 0 0 0 1 0
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.18
State's Heuristic value: 2
1 0 0 0 0 0
0 0 0 0 1 0
0 0 0 1 0 0

0 1 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.19
State's Heuristic value: 2
1 0 0 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
0 1 0 0 0 1
0 0 0 1 0 0
0 0 1 0 0 0
Sideways Move!

Step No.20
State's Heuristic value: 2
1 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 1 0 1
0 0 1 0 0 0
Sideways Move!

Step No.21
State's Heuristic value: 2
1 0 0 1 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 1 0 0 0
Sideways Move!

Step No.22
State's Heuristic value: 2
1 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 1 1 0 0
Sideways Move!

Step No.23
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0

Step No.24
State's Heuristic value: 1

1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.25
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
Sideways Move!

Step No.26
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.27
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0
Sideways Move!

Step No.28
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.29
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 1 1 0 0

Sideways Move!

Step No.30
State's Heuristic value: 1
1 0 1 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
Sideways Move!

Step No.31
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
Sideways Move!

Step No.32
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
Sideways Move!

Step No.33
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 1
0 1 0 0 0 0
Sideways Move!

Step No.34
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 1
Sideways Move!

Step No.35
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0

0 0 0 0 0 0
1 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
Sideways Move!

Step No.36
State's Heuristic value: 1
0 1 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
Sideways Move!

Step No.37
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 1
Sideways Move!

Step No.38
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 1
0 1 0 0 0 0
Sideways Move!

Step No.39
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 1 0 1
0 1 0 0 0 0
Sideways Move!

Step No.40
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.41
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.42
State's Heuristic value: 1
0 1 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.43
State's Heuristic value: 1
0 1 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.44
State's Heuristic value: 1
0 1 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.45
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.46
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0

0 0 0 1 0 1
0 1 0 0 0 0
Sideways Move!

Step No.47
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.48
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.49
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.50
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.51
State's Heuristic value: 1
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.52
State's Heuristic value: 1

1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.53
State's Heuristic value: 1
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.54
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.55
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.56
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.57
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 1
0 0 0 1 0 0

0 1 0 0 0 0
Sideways Move!

Step No.58
State's Heuristic value: 1
1 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.59
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.60
State's Heuristic value: 1
0 0 0 0 0 0
0 0 1 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.61
State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Success!
Total Steps Taken: 61

>>>
Initial Board
State's Heuristic value: 5
0 1 0 1 1 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

Step No.1

State's Heuristic value: 4
0 1 0 0 1 0
1 0 0 1 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

Step No.2
State's Heuristic value: 3
0 1 0 0 1 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 1 0 0

Step No.3
State's Heuristic value: 2
0 1 0 0 1 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0

Step No.4
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0

Step No.5
State's Heuristic value: 1
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.6
State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 1 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.7

State's Heuristic value: 1
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.8
State's Heuristic value: 0
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0

Success!
Total Steps Taken: 8

>>

Initial Board
State's Heuristic value: 4
0 0 1 0 0 0
0 0 0 0 1 1
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 4
0 0 0 0 0 0
0 0 0 0 1 1
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.2
State's Heuristic value: 2
0 1 0 0 0 0
0 0 0 0 1 1
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 1 0 0 0

Step No.3
State's Heuristic value: 2
0 1 0 0 0 1

0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.4
State's Heuristic value: 2
0 1 0 0 0 0
0 0 0 0 1 1
1 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.5
State's Heuristic value: 2
0 1 0 0 0 0
0 0 0 0 1 1
1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 0 1 0 0 0
Sideways Move!

Step No.6
State's Heuristic value: 1
0 1 0 0 0 0
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 0 1 0 0 0

Step No.7
State's Heuristic value: 1
0 1 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.8
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.9
State's Heuristic value: 1
0 0 0 0 0 0
0 0 1 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.10
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.11
State's Heuristic value: 1
0 0 0 0 0 0
0 0 1 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.12
State's Heuristic value: 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.13
State's Heuristic value: 1
0 0 0 0 0 0
0 0 1 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.14
State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1

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

Success!
Total Steps Taken: 14

>>

****
Hill Climbing Sideways Moves Algorithm
No. of Queens: 6
No. of Iterations: 485
Success and Fail Rates:---------
Success Rate: 85.56701 %
Failure Rate: 14.43299 %
Average no. of Steps When It Succeeds: 67
Average no. of Steps When It Fails: 212

3. Hill Climbing Random Restart Algorithm

>>
Initial Board
State's Heuristic value: 6
0 0 1 0 0 0
0 1 0 0 0 0
0 0 0 1 1 0
1 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0

Step No.20
State's Heuristic value: 5
0 0 1 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 0 0 1 0
0 0 0 0 0 0

Step No.21
State's Heuristic value: 3
0 0 1 0 0 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0

Step No.22
State's Heuristic value: 2
0 0 1 0 0 0
0 1 0 0 0 1
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 0 0

0 0 0 0 1 0

Step No.23
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
1 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 1 0
Random Restart!

>>
Initial Board
State's Heuristic value: 7
1 1 0 0 0 0
0 0 1 0 1 1
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 No.24
State's Heuristic value: 6
1 1 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 1

Step No.25
State's Heuristic value: 5
0 1 0 0 0 0
0 0 1 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 1

Step No.26
State's Heuristic value: 4
0 1 1 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 1

Step No.27
State's Heuristic value: 3
0 1 1 0 0 0
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 1 0 0

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

Step No.28
State's Heuristic value: 2
0 1 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 0 1

Step No.29
State's Heuristic value: 1
0 0 1 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 0 1
Random Restart!

>>
Initial Board
State's Heuristic value: 5
0 1 0 0 1 0
1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 0 1 0 0 1
0 0 0 0 0 0

Step No.30
State's Heuristic value: 4
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 1 1 0 0 1
0 0 0 0 0 0

Step No.31
State's Heuristic value: 2
0 0 0 0 1 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 1 0 0 0
0 0 0 0 0 0

Step No.32
State's Heuristic value: 1
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 1

0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
Random Restart!

>>
Initial Board
State's Heuristic value: 4
1 0 0 1 1 0
0 1 0 0 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

Step No.33
State's Heuristic value: 2
0 0 0 1 1 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 0 0 0

Step No.34
State's Heuristic value: 1
0 0 0 1 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 1
0 1 0 0 0 0
Random Restart!

>>
Initial Board
State's Heuristic value: 3
0 0 0 0 1 0
1 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 2
1 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0

Step No.2

State's Heuristic value: 1
1 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Step No.3
State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Success!
Total Steps Taken: 3

****
Hill Climbing Random Restart Algorithm
No. of Queens: 6
No. of Iterations: 445
Success and Fail Rates:---------
Success Rate: 100.0 %
Average no. of Steps When It Succeeds: 36
Average no. of Restarts When It Succeeds: 8

4. Hill Climbing Random Restart Algorithm with sideways move

>>>
Initial Board
State's Heuristic value: 5
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 1 0
0 1 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 3
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 1
0 0 1 0 0 0
0 0 0 0 1 0

Step No.2
State's Heuristic value: 3
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0

0 1 0 0 0 1
0 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.3
State's Heuristic value: 3
0 0 0 0 0 0
1 0 0 0 1 0
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.4
State's Heuristic value: 3
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.5
State's Heuristic value: 3
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 1 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.6
State's Heuristic value: 3
0 0 0 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 0 1
1 0 0 0 0 0
0 0 1 0 1 0
Sideways Move!

Step No.7
State's Heuristic value: 2
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 1 0 1 0

Step No.8
State's Heuristic value: 2

0 0 0 1 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
0 0 1 0 1 0
Sideways Move!

Step No.9
State's Heuristic value: 2
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 1 0
Sideways Move!

Step No.10
State's Heuristic value: 2
0 0 0 1 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
0 0 1 0 1 0
Sideways Move!

Step No.11
State's Heuristic value: 2
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.12
State's Heuristic value: 2
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 1 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.13
State's Heuristic value: 2
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 1 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0

Sideways Move!

Step No.14
State's Heuristic value: 2
0 0 0 1 0 1
0 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.15
State's Heuristic value: 2
0 0 0 1 0 0
0 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
1 0 0 0 0 1
0 0 1 0 0 0
Sideways Move!

Step No.16
State's Heuristic value: 2
0 0 0 1 0 1
0 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 1 0 0 0
Sideways Move!

Step No.17
State's Heuristic value: 1
0 0 0 1 0 1
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0

Step No.18
State's Heuristic value: 1
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 1
Sideways Move!

Step No.19
State's Heuristic value: 0
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0

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

Success!
Total Steps Taken: 19

>>
Initial Board
State's Heuristic value: 5
0 1 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 1 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 5
0 1 0 0 0 1
0 0 1 0 1 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.2
State's Heuristic value: 5
0 1 0 0 0 1
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 1 0
0 0 0 0 0 0
Sideways Move!

Step No.3
State's Heuristic value: 5
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
1 0 0 1 1 0
0 0 0 0 0 0
Sideways Move!

Step No.4
State's Heuristic value: 5
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 1 0
0 0 0 0 0 1
Sideways Move!

Step No.5
State's Heuristic value: 4
0 1 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 0 0
0 0 0 0 0 1

Step No.6
State's Heuristic value: 3
0 1 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
1 0 0 0 0 1

Step No.7
State's Heuristic value: 2
0 1 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
1 0 0 0 0 0

Step No.8
State's Heuristic value: 2
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
1 0 0 0 0 0
Sideways Move!

Step No.9
State's Heuristic value: 1
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0

Step No.10
State's Heuristic value: 1
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 1 0 1
0 1 0 0 0 0
0 0 0 0 0 0
Sideways Move!

Step No.11
State's Heuristic value: 1
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 0 0
Sideways Move!

Step No.12
State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Success!
Total Steps Taken: 12

>>
Initial Board
State's Heuristic value: 6
1 1 0 0 0 0
0 0 1 0 1 0
0 0 0 1 0 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0

Step No.1
State's Heuristic value: 4
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 1 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0

Step No.2
State's Heuristic value: 4
1 0 0 0 0 0
0 0 1 0 1 0
0 0 0 0 0 0
0 1 0 0 0 1
0 0 0 0 0 0
0 0 0 1 0 0
Sideways Move!

Step No.3
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 1 0

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

Step No.4
State's Heuristic value: 2
1 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 1 0 0
0 0 0 0 1 0
Sideways Move!

Step No.5
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 1 0
Sideways Move!

Step No.6
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 1 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0
Sideways Move!

Step No.7
State's Heuristic value: 2
1 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 1 0
Sideways Move!

Step No.8
State's Heuristic value: 2
0 0 0 0 0 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 1 0
Sideways Move!

Step No.9

State's Heuristic value: 0
0 0 0 0 1 0
0 0 1 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 1 0 0
0 1 0 0 0 0

Success!
Total Steps Taken: 9

****
Hill Climbing Random Restart Algorithm with sideways move
No. of Queens: 6
No. of Iterations: 466
Success and Fail Rates:---------
Success Rate: 100.0 %
Average no. of steps when it succeeds: 92
Average no. of Restarts when it succeeds: ~ 1
Continue algorithm? Y or N
N

Process finished with exit code 0

10. References:

1. Lecture slides by Prof. Dewan Ahmed