	Finolex Academy of Management and Technology, Ratnagiri
	Department of Information Technology

Subject name: Intelligent System Labs				Subject Code: BEITC703
Class	BE IT	Semester – VII (CBGS)		Academic year: 2019-20
Name of Student	**Kazi Jawwad A Rahim**		**QUIZ Score :**	
Roll No	**29**	Assignment/Experiment No.		03
Title: **To implement 8 puzzle problem with Heuristic function using Hill climbing.**				

1. Course objectives applicable: COB2. Understand the different informed search techniques to solve the different AI problems like 8 puzzle

2. Course outcomes applicable:
CO2 –Solve the problems based on informed searching techniques.

3. Learning Objectives:
1. To understand different informed search techniques.
2. To understand 8 puzzle problem
3. To program 8 puzzle problem using hill climbing algorithm.
4. To get the output this will calculate the elapsed time and heuristic function value to get goal state.

4. Practical applications of the assignment/experiment: sliding puzzle game application

5. Prerequisites:
1. To learn the use of intelligent agents in informed search.
2. To understand the programming methodology for 8 puzzle problem.

6. Hardware Requirements:
1. PC with minimum 2GB RAM

7. Software Requirements:
1. Windows installed
2. JDK/Net beans

8. Quiz Questions (if any): (Online Exam will be taken separately batchwise, attach the certificate/ Marks obtained)
1. How informed search strategies are defined?
2. In Best First search node is selected for expansion on which basis?
3. Which is the type of hill climbing?
4. Which is not drawback of hill climbing?

9. Experiment/Assignment Evaluation:

Sr. No.	**Parameters**		**Marks obtained**	**Out of**
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used -			6
2	Neatness/presentation			2
3	Punctuality			2
Date of performance (DOP)		**Total marks obtained**		**10**
Date of checking (DOC)		**Signature of teacher**		

11. Precautions:

1. Generate all possible next steps of the solution.
2. Set the directions in which tiles should be moved.

12. Installation Steps / Performance Steps –

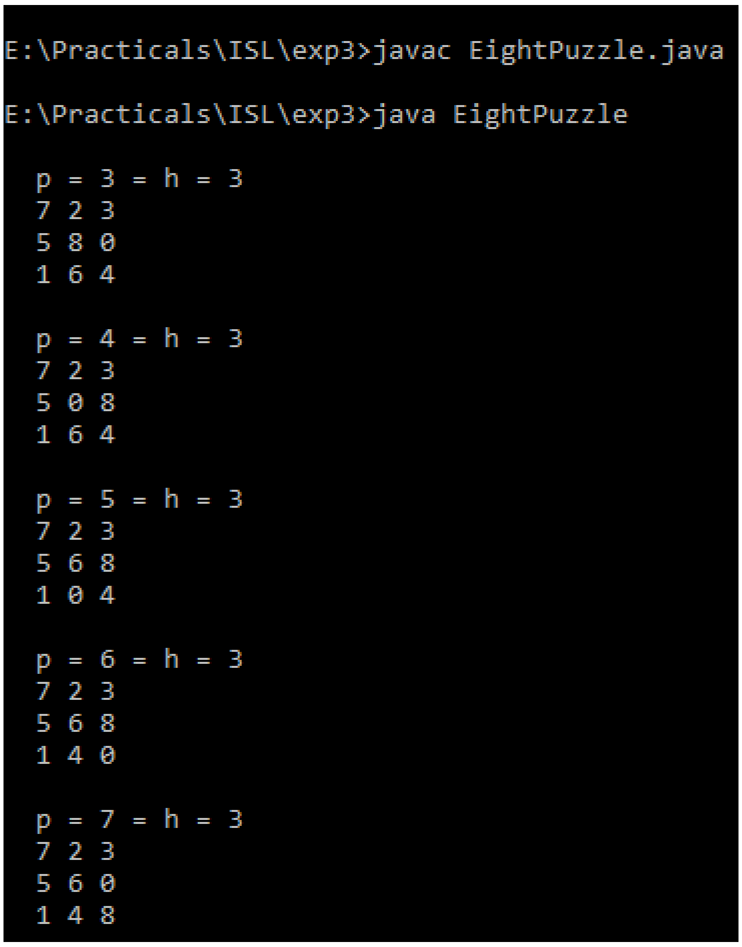
```
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

public class EightPuzzle {


   static final byte [] goalTiles = { 1, 2, 3, 4, 5, 6, 7, 8, 0 };//Tiles for completed puzzle
   final PriorityQueue <State> queue = new PriorityQueue<State>(100, new Comparator<State>()
{
       @Override
       public int compare(State a, State b) {
          return a.priority() - b.priority();
       }
   });

   //The closed state set.
   final HashSet <State> closed = new HashSet <State>();

   class State
{
       final byte [] tiles;    //Tiles left to right,top to bottom
       final int spaceIndex;   //Index of spaces zero(blank slide)
       final int g;            //No of moves from start
       final int h;            //Heuristic value i.e.differencefrom goal state
       final State prev;       //previous state


       int priority()
          {
          return g + h;
       }


       State(byte [] initial)    //Build starting state
          {
          tiles = initial;
          spaceIndex = index(tiles, 0);
          g = 0;
          h = heuristic(tiles);
          prev = null;
       }


       State(State prev, int slideFromIndex)   //Build successor to previous by sliding from current
state
           {
          tiles = Arrays.copyOf(prev.tiles, prev.tiles.length);
          tiles[prev.spaceIndex] = tiles[slideFromIndex];
```

```
        tiles[slideFromIndex] = 0;
        spaceIndex = slideFromIndex;
        g = prev.g + 1;
        h = heuristic(tiles);
        this.prev = prev;
    }


    boolean isGoal()    //Will return true value if goal test succeed
         {
        return Arrays.equals(tiles, goalTiles);
    }

    //Successor moves to south,north,east aand west sides.
    State moveS() { return spaceIndex > 2 ? new State(this, spaceIndex - 3) : null; }
    State moveN() { return spaceIndex < 6 ? new State(this, spaceIndex + 3) : null; }
    State moveE() { return spaceIndex % 3 > 0 ? new State(this, spaceIndex - 1) : null; }
    State moveW() { return spaceIndex % 3 < 2 ? new State(this, spaceIndex + 1) : null; }



    void print()   //Printing the current state
        {
        System.out.println("  p = " + priority() + " = h = " + h);
        for (int i = 0; i < 9; i += 3)
            System.out.println("  " + tiles[i] + " " + tiles[i+1] + " " + tiles[i+2]);
    }


    void printAll()  //Print the solution chain fromstart state
         {
        if (prev != null) prev.printAll();
        System.out.println();
        print();
    }

    @Override
    public boolean equals(Object obj)
        {
        if (obj instanceof State)
            {
            State other = (State)obj;
            return Arrays.equals(tiles, other.tiles);
        }
        return false;
    }

    @Override
    public int hashCode()
        {
        return Arrays.hashCode(tiles);
    }
}
```

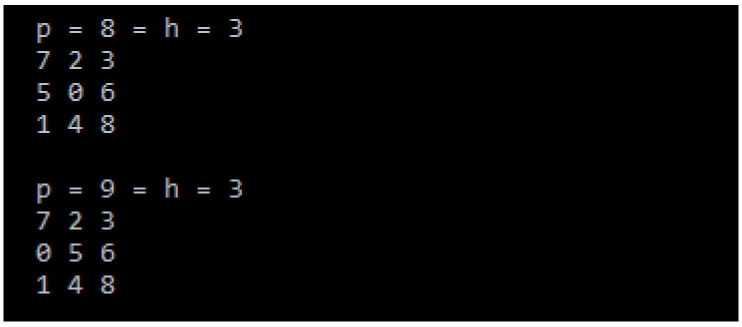
```
    void addSuccessor(State successor) //adds the valid successor i.e.non null and not closed
          {
       if (successor != null && !closed.contains(successor))
          queue.add(successor);
           }


    void solve(byte [] initial) //Running  the solver
          {

       queue.clear();
       closed.clear();


       long start = System.currentTimeMillis();   //capturing the systems time elapsed to solve


       queue.add(new State(initial));

       while (!queue.isEmpty()) {


          State state = queue.poll();


          if (state.isGoal())
             {
             long elapsed = System.currentTimeMillis() - start;
             state.printAll();
             System.out.println("  Elapsed (ms) = " + elapsed);
             return;
          }


          closed.add(state);  //To make sure that we are not evisiting states


          addSuccessor(state.moveS());
          addSuccessor(state.moveN());
          addSuccessor(state.moveW());
          addSuccessor(state.moveE());
       }
    }


    static int index(byte [] a, int val)  //Returns the index of value in given byte of array
    {
       for (int i = 0; i < a.length; i++)
          if (a[i] == val) return i;
       return -1;
    }


    static int manhattanDistance(int a, int b) //Returns manhatan distance between tiles with  indices
a and b
```

```
    {
        return Math.abs(a / 3 - b / 3) + Math.abs(a % 3 - b % 3);
    }

        static int heuristic(byte [] tiles)
    {
        int h = 0;
        for (int i = 0; i < tiles.length; i++)
            if (tiles[i] != 0)
                h = Math.max(h, manhattanDistance(i, tiles[i]));
        return h;
    }
    public static void main(String[] args) {


        byte [] initial = { 7, 2, 3, 5, 8, 0, 1, 6, 4 };//Initial state is provided to system


        //byte [] initial = { 1, 4, 2, 3, 0, 5, 6, 7, 8 };

        new EightPuzzle().solve(initial);
    }
}
```

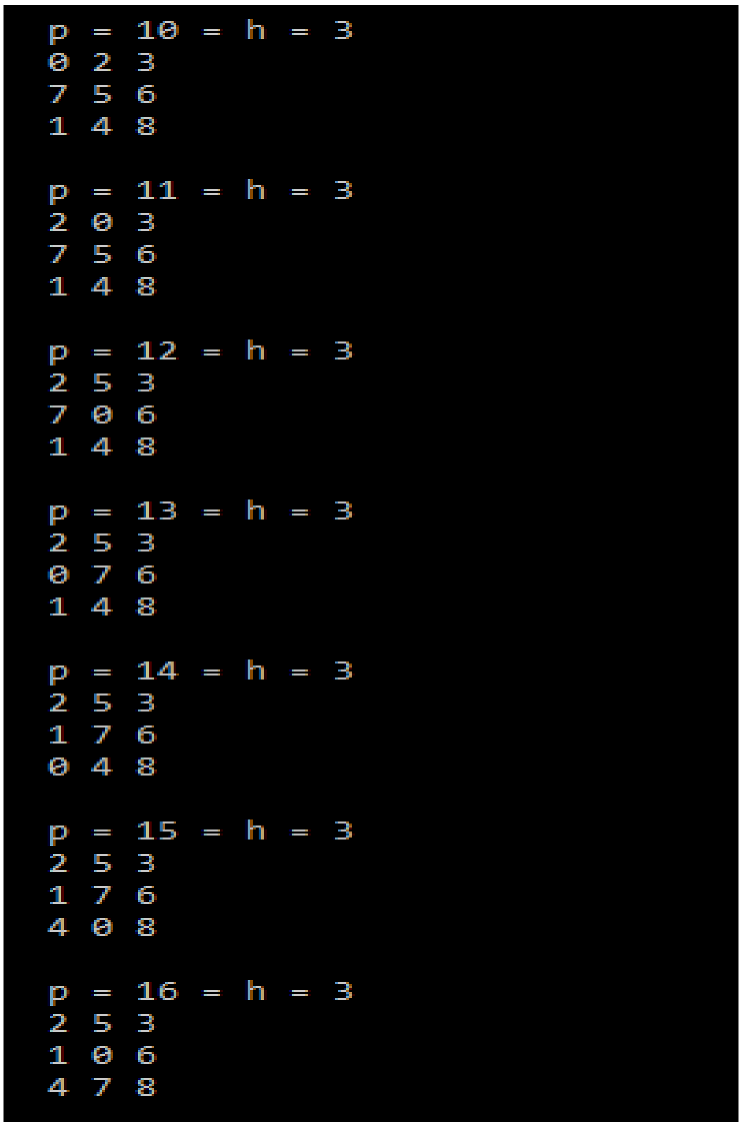
14. Results:



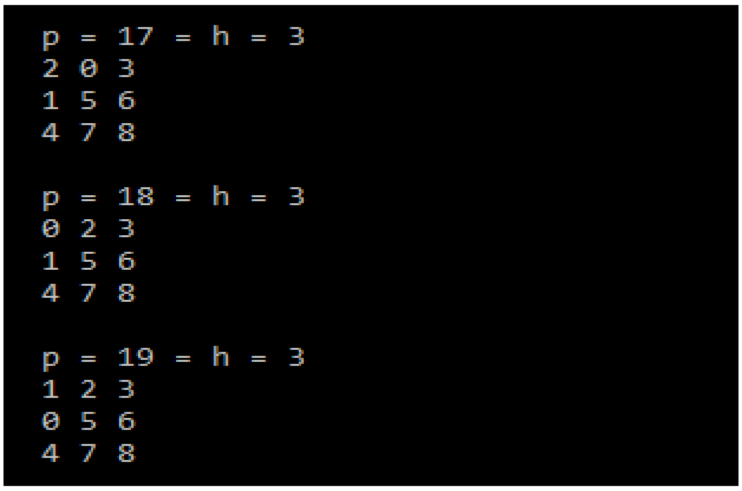
```
E:\Practicals\ISL\exp3>javac EightPuzzle.java

E:\Practicals\ISL\exp3>java EightPuzzle

  p = 3 = h = 3
  7 2 3
  5 8 0
  1 6 4

  p = 4 = h = 3
  7 2 3
  5 0 8
  1 6 4

  p = 5 = h = 3
  7 2 3
  5 6 8
  1 0 4

  p = 6 = h = 3
  7 2 3
  5 6 8
  1 4 0

  p = 7 = h = 3
  7 2 3
  5 6 0
  1 4 8
```



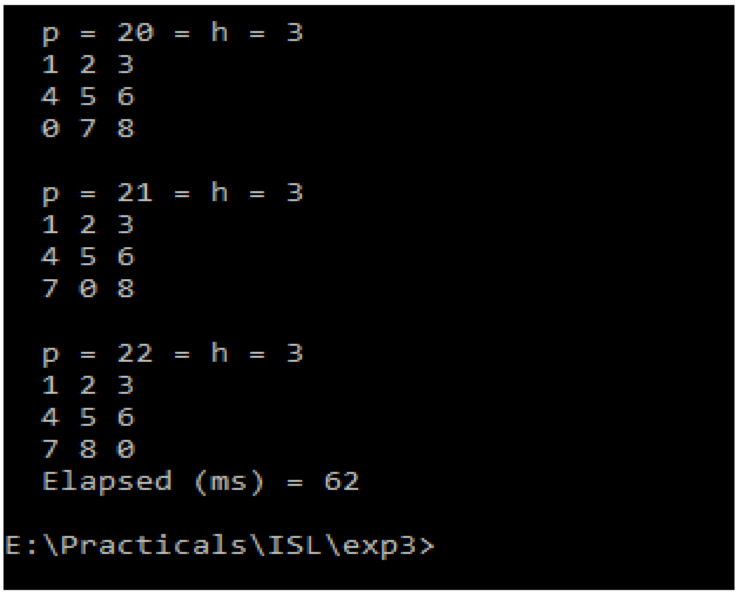
```
p = 8 = h = 3
7 2 3
5 0 6
1 4 8

p = 9 = h = 3
7 2 3
0 5 6
1 4 8
```



```
p = 10 = h = 3
0 2 3
7 5 6
1 4 8

p = 11 = h = 3
2 0 3
7 5 6
1 4 8

p = 12 = h = 3
2 5 3
7 0 6
1 4 8

p = 13 = h = 3
2 5 3
0 7 6
1 4 8

p = 14 = h = 3
2 5 3
1 7 6
0 4 8

p = 15 = h = 3
2 5 3
1 7 6
4 0 8

p = 16 = h = 3
2 5 3
1 0 6
4 7 8
```



```
p = 17 = h = 3
2 0 3
1 5 6
4 7 8

p = 18 = h = 3
0 2 3
1 5 6
4 7 8

p = 19 = h = 3
1 2 3
0 5 6
4 7 8
```



```
  p = 20 = h = 3
  1 2 3
  4 5 6
  0 7 8

  p = 21 = h = 3
  1 2 3
  4 5 6
  7 0 8

  p = 22 = h = 3
  1 2 3
  4 5 6
  7 8 0
  Elapsed (ms) = 62

E:\Practicals\ISL\exp3>
```

15. Learning Outcomes Achieved

1. Understood the concept of informed search.
2. Used heuristic function to solve 8 puzzle problem using hill climbing algorithm.

16. Conclusion:

1. **Applications of the studied technique in industry**
   a. Heuristic functions are used to find the solution path based on path cost which is used to develop game application like sliding puzzle.
2. **Engineering Relevance**
   a. Such algorithms are very useful in searching techniques where problems are more complex.
3. **Skills Developed**
   a. Implementation of hill climbing algorithm to solve 8 puzzle problem.

17. References :

[1] G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.) “Handbuch der künstlichen Intelligenz” Oldenbourg Verlag, 2003, Fourth edition

• [2] Turing, A. "Computing Machinery and Intelligence", Mind LIX (236): 433–460, Ocotober, 1950.

• [3] Aristotle “On Interpretation”, 350 B.C.E, see: http://classics.mit.edu/Aristotle/interpretation.html

• [4] Newell, A., Simon, H.A. “Human Problem Solving” Englewood Cliffs, N.J.: Prentice Hall, 1972.