|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Finolex Academy of Management and Technology, Ratnagiri** | | | | | | | | | |
| **Department of Information Technology** | | | | | | | | | |
| Subject name: Intelligent Systems Labs | | | | | | | | Subject Code: BEITC703 | | | |
| Class | | BE IT | | Semester – VII (CBGS) | | | | Academic year: 2019-20 | | | |
| Name of Student | |  | | | | | **QUIZ Score :** | | | | |
| Roll No | |  | | | Assignment/Experiment No. | | | | | 04 | |
| Title:  **To implement 8 puzzle problem with Heuristic function using Best first search.** | | | | | | | | | | | |
|  | | | | | | | | | | | |
| **1. Course objectives applicable:**  **COB2**.Understand the different informed searching techniques to solve the different AI problems on the basis heuristic functions. | | | | | | | | | | | |
| **2. Course outcomes applicable:**  **CO2** –Solve the problems based on heuristic functions used for searching. | | | | | | | | | | | |
| **3. Learning Objectives:**   1. To understand concept of informed search. 2. To understand heuristic functions. 3. To program for 8 puzzle problem by using best first search algorithm. 4. To get the output which will calculate heuristic function values ad returns the steps towards goal state. | | | | | | | | | | | |
| **4. Practical applications of the assignment/experiment: complex problems used for gaming like chess and 8 puzzle.** | | | | | | | | | | | |
| **5. Prerequisites**:   1. To learn the use of intelligent agents in informed search. 2. To understand the programming methodology for best first search using heuristic functions. | | | | | | | | | | | |
| **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 batch wise, attach the certificate/ Marks obtained)**   1. What do you mean by heuristic functions? 2. What is Best First Search algorithm? 3. What is fringe? 4. What is space complexity of best first search algorithm? | | | | | | | | | | | |
|  | | | | | | | | | | | |
| **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. Find heuristic function values h(n) and general function f(n).

2. Calculate both functions to find the goal state.

**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, 8, 0, 4, 7, 6, 5 };

// A\* priority queue.

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>();

// State of the puzzle including its priority and chain to start state.

class State

{

final byte [] tiles; // Tiles left to right, top to bottom.

final int spaceIndex; // Index of space (zero) in tiles

final int g; // Number of moves from start.

final int h; // Heuristic value (difference from goal)

final State prev; // Previous state in solution chain.

// A\* priority function (often called F in books).

int priority()

{

return g + h;

}

// Build a start state.

State(byte [] initial)

{

tiles = initial;

spaceIndex = index(tiles, 0);

g = 0;

h = heuristic(tiles);

prev = null;

}

// Build a successor to prev by sliding tile from given index.

State(State prev, int slideFromIndex)

{

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;

}

// Return true iif this is the goal state.

boolean isGoal()

{

return Arrays.equals(tiles, goalTiles);

}

// Successor states due to south, north, west, and east moves.

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; }

// Print this state.

void print()

{

System.out.println(" p = " + priority() + " = g+h = " + g + "+" + h);

for (int i = 0; i < 9; i += 3)

System.out.println(" " + tiles[i] + " " + tiles[i+1] + " " + tiles[i+2]);

}

// Print the solution chain with start state first.

void printAll()

{

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);

}

}

// Add a valid (non-null and not closed) successor to the A\* queue.

void addSuccessor(State successor) {

if (successor != null && !closed.contains(successor))

queue.add(successor);

}

// Run the solver.

void solve(byte [] initial) {

queue.clear();

closed.clear();

// Click the stopwatch.

long start = System.currentTimeMillis();

// Add initial state to queue.

queue.add(new State(initial));

while (!queue.isEmpty()) {

// Get the lowest priority state.

State state = queue.poll();

// If it's the goal, we're done.

if (state.isGoal()) {

long elapsed = System.currentTimeMillis() - start;

state.printAll();

System.out.println(" Elapsed (ms) = " + elapsed);

return;

}

// Make sure we don't revisit this state.

closed.add(state);

// Add successors to the queue.

addSuccessor(state.moveS());

addSuccessor(state.moveN());

addSuccessor(state.moveW());

addSuccessor(state.moveE());

}

}

// Return the index of val in given byte array or -1 if none found.

static int index(byte [] a, int val) {

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

if (a[i] == val) return i;

return -1;

}

// Return the Manhatten distance between tiles with indices a and b.

static int manhattanDistance(int a, int b) {

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

}

// For our A\* heuristic, we just use max of Manhatten distances of all tiles.

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) {

// This is a harder puzzle than the SO example

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

// This is taken from the SO example.

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

new EightPuzzle().solve(initial);

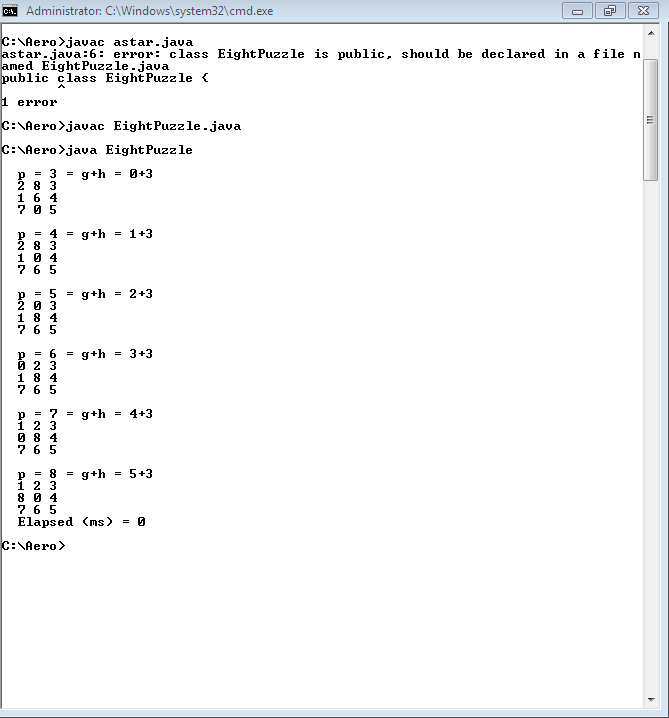
}

}

**13. Observations**

1. The output will give the sequence of steps to find the solution path or goal node.

**14. Results:**

****

**15. Learning Outcomes Achieved**

1. Understanding the concept of informed search.
2. Understanding the eight puzzle problem solved by informed search technique.

**16. Conclusion:**

1. **Applications of the studied technique in industry**
   1. Best first algorithm used to develop intelligent systems which will be used to develop games like 8 puzzle.
2. **Engineering Relevance** 
   1. Such algorithms are very useful in searching techniques where number of solutions are more than one and complex.
3. **Skills Developed**
   1. Implementation of Best first search for 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

• [5] Newell, A. “The Knowledge Level”, AI Magazine 2 (2), 1981, p. 1-20.