**CSC412/512 – Intelligent Systems**

**HW2 – Due Monday, February 12, 11:59 PM**

The purpose of this assignment is to implement a couple of graph-based search methods. The problem to be solved is the 8-tile sliding puzzle.

You can implement your code in Java, C++ or Python. You are not to use any canned search algorithms; you must implement the search portion yourself. I have implemented much of the code for you, so you should just be able to concentrate on the algorithms themselves. My code is in Java but you can use it as a starting point to implement in another language if you wish.

Here are the classes that I am providing with this assignment:

**SGProblem** is a class that defines, in general terms, the problem to be solved. It is written to fit the concept that the author uses in the text. It is complete, and probably doesn’t need any changes.

**SGAction** is a very simple class that just wraps a string for the four possible moves in the sliding puzzle game: “U” to move the blank spot up, etc. It is complete and doesn’t need any changes.

**SGSolution** encapsulates a sequence of actions that will solve the problem (when the search is done). It is complete and doesn’t need any changes.

**SGNode** represents a single node in the graph representation of the problem space. It is as discussed on pages 78 and following. It is complete and doesn’t need any changes.

**SGState** is where the specifics of the 8-tile sliding puzzle are contained (mostly). The current state of the board is represented by this class. It has lots of methods to support the needs of the game. It is complete and doesn’t need any changes.

**SearchGraph** (copied below) is where you will be doing your work. It has two important methods that are incomplete: breadthFirstSearch(SGProblem p) and uniformCostSearch(SGProblem p). I want you to write the code for these two methods, to implement the pseudocode listed on pages 82 and 84.

To run the program, edit line 10 of SearchGraph to set “TESTDATA” to a value from 0 to 4. Values 1 through 4 will create a new puzzle with small sets of steps necessary to solve the problem. The value 0 will create a random puzzle (which may take a long time to solve!). I advise you to use values 1 and 2 to debug your code. To run the solution, edit the main() method in SearchGraph to use one of the two search methods I have asked you to write, then see what it does.

NOTE 1: I wrote all of this fairly quickly, but if you think you have found a problem in my code, CHECK WITH ME BEFORE ASSUMING IT’S WRONG AND CHANGING IT!

NOTE 2: I emphasize again, you are NOT to download someone else’s code for solving this problem – whether source code or a package. You are to implement the algorithms from the book and experiment with them.

When you are done, put all of the code for your SearchGraph class, along with the TEXT (no screenshots) from running your program on a few test setups, into a single Word file and submit via Blackboard.

// SearchGraph class Creed Jones CBU CSC512 SP18 Jan 28, 2018

// this class supports solving the 8-puzzle problem

import java.util.LinkedList;

import java.util.PriorityQueue;

import java.util.Iterator;

public class SearchGraph {

public static final int TESTDATA = 4;

public SGSolution breadthFirstSearch(SGProblem p) { // as discussed on page 82

LinkedList<SGNode> frontier = new LinkedList<SGNode>();

LinkedList<SGState> explored = new LinkedList<SGState>();

{{{ YOUR CODE GOES HERE… }}}

}

public SGSolution uniformCostSearch(SGProblem p) { // as discussed on page 84

final int INITIALSIZE = 1000; // arbitrary - will resize if needed

PriorityQueue<SGNode> frontier =  
 new PriorityQueue<SGNode>(INITIALSIZE, new SGNode.SortByCost());

LinkedList<SGState> explored = new LinkedList<SGState>();

{{{ YOUR CODE GOES HERE… }}}

}

public static void main(String[] args) {

SearchGraph sg = new SearchGraph();

SGSolution soln = sg.uniformCostSearch(new SGProblem());

// SGSolution soln = sg.breadthFirstSearch(new SGProblem());

System.out.println("Solution is " + soln.getLength() + " long;");

System.out.println(soln.toString());

}

}