CS540 – Assignment 3

CS540 – Fall 2015 – Sections 2 & 4
University of Wisconsin, Madison

Submission and Deadlines:

You will submit two separate files through Moodle for this assignment: (1) a .pdf file with your written answers to the theory questions, and (2) a .java file with your code implementation for the practical problems. Both of these files shall be submitted no later than **Noon on Friday, November 13th** to receive full credit. Late submissions will be penalized 10% per 24 hours that they are received late. After five days (120 hours), late submissions will no longer be accepted for credit.

Collaboration Policy:

This entire assignment shall be completed individually. While you are encouraged to discuss relevant concepts and algorithms with classmates and TAs, you shall only use different examples and problems from those assigned below. Academic misconduct on this assignment includes, but is not limited to: sharing, copying, and failing to protect the privacy of your answers and code fragments with respect to the assignment described below.

CSP and Propositional Logic (Theory)

(1-3). Consider a 3x3 Sudoku puzzle where each position in the grid can be assigned the value: 1, 2, or 3. The values in each column and each row of this grid must be all different. The puzzle is initialized with a 2 in the middle column of the top row. The rest of the grid is labeled with letters to help you reference them in the following problems:

A	**2**	C
D	E	F
G	H	I

1. List the binary constraints among the grid positions in the right-most column of this grid: between C, F, and I.

2. If a backtracking-search is run on this problem using the minimum remaining value heuristic, then which remaining grid position will be assigned a value next. Assume that the highest degree heuristic is used as a first tie-breaker, and that the alphabetical order of the position labels is used as a second tie-breaker.

3. Assume that a 3 is added to the grid position G after the 2, and that a forward checking inference procedure was run after each of these two variable assignments. Which of the remaining grid positions will this forward checking procedure be able to infer a value for after this 3 is assigned?

4. Again assume that a 3 is added to position G in the grid. What will this grid look like after the AC-3 algorithm (from in Figure 6.3 of your text) is then run? Draw the single remaining value for each grid position that has only one possible value left in its domain. And draw the

letter label on grid positions with more than one value left in their domain.

5. Which of the following logical statements best capture the relationships described in this English sentence: You should attend class (AC) if you want a good grade (GG), but only after reviewing your notes (RN).

a. AC => (GG && RN)

b. RN && (GG =>AC)

c. (RN && GG) => AC

d. GG => (AC && RN)

6. Convert the following statement into Conjunctive Normal Form: (X => !Y) <=> Z

7. List all of the clauses that will result from performing resolution on the following statement. Do not include clauses that are always true, do not include duplicates of any clause, and alphabetize both the literals in each individual clause and the clauses. For example, the clauses (!X || Y || X) && (X || Y) && (Y || X) && (Z || Y || X), should be reported as: (X || Y) && (X || Y || Z). Here is the statement that you should resolve:

(A || B || !C) && (!A || C) && (B || C) && !B

Heuristic A* Search (Practical)

0. For this portion of the assignment, you will be implementing an A* Heuristic Search on a map from http://OpenStreetMap.org. Your implementation will be exclusively written into the provided class skeleton FIRSTNAME_LASTNAME_AStar.java. After replacing the FIRSTNAME_LASTNAME portion of this filename with your own name, you will need to update the classname within this file, and the two references used to instantiate this class in Main.java: one to declare the reference variable and one to instantiate an instance of your class. Your final solution should run without any other changes being made to Main.java or Map.java. DO NOT include any package statements, and DO NOT call exit from any of the methods that you are implementing.

   This code utilizes Processing version 3.0. The core.jar file for this version of the library is available through the moodle page for this assignment. It should be added to your project and to your project's build path to run the provided code.

   Data
   For this assignment, you will be searching through map data from http://OpenStreetMap.org. You are provided with a sample map of Madison, WI, but are encouraged to experiment with other smaller maps for testing purposes. The provided Map class is able to read in the points and streets from one of these map files, and does some filtering of bicycle and walking paths in the process. The result is a network of Map.Point objects that each contain their own x and y position along with an ArrayList of neighboring points called neighbors. Your code will be able to crawl through this network of Map.Point objects by traversing through these lists of neighbor

references.

1. Within your AStar search class, there are two ArrayLists for you to keep track of the points that you have explored, and are considering exploring next according to their priority. The type of the objects in each of these lists is the SearchPoint class that is defined within your AStar class. This SearchPoint class is yours to implement and extend as you see fit. At a minimum you will need to keep track of the mapPoint that each SearchPoint represents reaching. Additionally you will need to implement the g(), h(), and f() methods to calculate the incurred cost to reach this point, the estimated remaining cost, and the resulting priority for exploring this point in the future. In addition to these methods, you will override the compareTo() and equals() methods of this class, so that you can more easily sort your frontier, and check whether your frontier or explored ArrayLists contain a SearchPoint that corresponds with a given Map.Point.

2. The bulk of your A* Search algorithm will be implemented between two methods: the constructor of your Astar class will initialize all of your class member variables, and add the start point from the provided map to your frontier to begin the search. This constructor also receives the type of heuristic that should be used via the parameter H. For this parameter: 0 = zero heuristic which evaluates to zero for every state, 1 = manhattan distance heuristic which returns the sum of the x and y offesets between the point in question and the goal point, and 2 = the standard Euclidean L2 straight-line distance between the point in question and the goal point. Once this class has been instantiated, the Main class will repeatedly call it's exploreNextNode() method. Every time this method is called, it should explore the single highest priority SearchPoint from the frontier. This method may also need to do some work to help keep track of whether the search has completed, and if so what point the solution passes through. There are reported back to main by other methods, but some may find it helpful to update these kinds of state changes from this exploreNextNode() method.

3. The other required methods in your AStar class are used to help you monitor and verify the correctness of your search as it proceeds and is completed. The getFrontier() and getExplored() methods return an ArrayList of the Map.Points corresponding to the SearchPoints in each of your ArrayLists. These are used by main to help you monitor how your search progresses. After you have found the solution or exhausted all possible chances of finding one, isComplete() should return true. After this happens, the getSolution() method should return an ArrayList of the Map.Points that your solution path passes through while traversing from the start point to the end point. These points must be returned in the correct order.

4. Congratulations. Now that you have implemented each of the assigned methods, you are left only with the task of testing and validating the code that you have written. You are encouraged to try running your code on some smaller .osm map files to help validate the results of your search. The provided visualization and GUI code should help you in this endeavor. Not only can you compare the number of nodes explored before reaching a solution with each of the three heuristics, but you should also be able to find a way of causing the inadmissible heuristic to find sub-optimal solution.