Computer Science 300 – Spring 2014
Assignment #2

100 points **Due: Monday, 3/24/14 at 11:59PM**

The code for both parts of this assignment is available in the file `A2.zip` on D2L.

1. (50 points) Recall the code we used in Assignment 1, Part 2. We will use that framework again, to add a couple more classes. One more example problem has been added, the Eight-puzzle problem described in the lecture notes, and partially implemented in the files `EightPuzzle.java` and `EightPuzzleNode.java.`

   Provide code to answer the following questions:

   a) (30 points) **A* Implementation**: Implement the complete version of the A* algorithm, which I demonstrated in class, in a class called `AStar`. You MUST use the Java API `PriorityQueue` implementation for the open list, and provide a comparator function to tell it how to order the priority queue. To promote nodes on the priority queue, you will need to use an iterator to delete a node and then add that modified node back on to the queue. To test your implementation of A*, run it on the `Romania` problem. Please follow the API/template we have used to write other algorithms such as `BFS_DD`.

   b) (20 points) **Heuristic Implementation**: Implement the Manhattan distance heuristic (block distance to goal) and Hamming distance heuristic (number of misplaced tiles) for the 8 Puzzle. Your implementation must be done in the `getHeuristicValue()` method in the `EightPuzzle.java` file. This method is stubbed out for you. (You can certainly call other utility methods you add.) A value of 1 in the command line arguments should run the Hamming distance heuristic and a value of 2 should run the Manhattan distance heuristic. Test your implementation of A* on this problem with both heuristics and note which heuristic, if any, gives better performance.

2. (50 points) Recall that Sudoku is a game played on a square grid of 9 large blocks, each containing its own 3x3 grid of cells. The rules are simple:
   - each cell contains a number from 1-9
   - the number in a given cell (i, j) cannot match the number in any other cell in row i or column j
   - each block can only contain one entry of each number from 1-9

   Some cells are initially filled in and the goal is to find the unique assignment of numbers to the remaining cells that satisfies the above rules. This game can be seen as a straightforward system of constraints.

   So, for this problem, you will implement a solver for Sudoku puzzles using CSP strategies. Again, you will modify an existing code base for this problem. The relevant files are:

- `Variable.java` : A class to represent a variable in a CSP.
- `Value.java` : A class that represents a single value a variable can take.
- `CBNode.java` : A class that represents a node used during chronological backtracking. A node wraps a variable.
- `NEConstraint.java` : A class to represent a constraint in a Sudoku puzzle. Since all constraints are <> constraints, this is a binary <> constraint.
- `Arc.java` : A class to represent a directional arc constraint in a CSP. Every binary constraint can be broken up into two arcs.
- `SudokuProblem.java` : An class to capture a CSP representation for a Sudoku puzzle and associated algorithms to solve it.
- `SudokuUtil.java` : This is a class that handles reading and writing the Sudoku boards to text files for you. The board is represented as a two dimensional array of integers.
- `Driver.java` : This is a test script that will be used to test your code. You may want to modify it for testing/debugging of your implementation, but make sure that the code you submit works correctly with an unmodified copy of Driver.

You will do all your work in the `SudokuProblem` class. In particular, you will

a) (40 points) Implement the AC3 arc consistency algorithm demonstrated in class in the stubbed out method `AC3()`
b) (10 points) Implement the Minimum-Remaining-Value (MRV) heuristic to choose the next variable to assign using chronologial backtracking. This method is stubbed out for you in the method `MRVHeuristic()`.

You can add other helper methods to the `SudokuProblem` class as needed, but this is the only file you should have to modify.

To run your program, run the `main` method in the `Driver` class. Four sample sudoku puzzles and their solutions are enclosed in the same directory. The driver will attempt to solve these four puzzles by calling `AC3()` and `chronologicalBacktrack()`. It will then test the answer with the supplied solutions. You may comment out lines in the `Driver` class, as appropriate, when testing. Submit your `SudokuProblem.java` file into the D2L drop box. This is the only file to submit for this problem.

Submission
You may do this assignment with a partner, if you choose. If so, it is expected that you both understands all parts of the code you submit, and you will both receive identical grades.

Submit the following files, zipped up in a single folder called `Assignment2.zip` to the `Assignment2` drop box on D2L by the deadline specified:

1) A `Readme` file with the following information in it:
   a) Name of partner, if any:

b) Statement that you understood all the code that was submitted, even if done by your partner: (Yes/No)

c) Question 1(a):
   - Successfully completed: Yes/No/Yes but...

d) Question 1(b):
   - Successfully completed: Yes/No/Yes but...

e) Question 2(a):
   - Successfully completed: Yes/No/Yes but...

f) Question 2(b):
   - Successfully completed: Yes/No/Yes but...

2) `AStar.java` for Question 1(a).

3) `EightPuzzle.java` for Question 1(b).

4) `SudokuProblem.java` for Question 2(a) and 2(b).