Homework #2: Chapters 3 and 5

The following exercises are due at the beginning of class on **Tuesday, March 1.** This assignment includes an extra-credit assignment worth up to an additional 20 points.

1. *[15 points]* Consider the road map of Romania in Fig. 3.2 on p. 68 of the text book. Use Greedy Best-First search to find a path from Dobreta to Bucharest, where step costs are as labeled on the map and the values for the heuristic function are given by Fig. 3.22 on p. 93. Show your search tree, including the h(n) value for each node, and label each node with the order in which it is expanded (note, this may be different from the order it is generated). In order to reduce unnecessary search, you should use graph-based search.
2. *[15 points]* Now repeat the exercise above, but use A* instead of greedy best-first. Show your search tree, complete with f(n), g(n) and h(n) values for each node, and label each node with the order in which it is expanded. Once again, use a graph search approach, but remember that if a generated node has a lower f(n) than a frontier node with an identical state, then it must replace that node on the frontier.
3. *[20 points]* Use A* to solve the 8-puzzle with the initial and goal states shown below. Assume that your path cost is 1 per move and that your heuristic function is h_2, the Manhattan distance of all tiles from their correct placement (note, the blank does not count as a tile). Show your search tree, complete with f(n), g(n) and h(n) values for each node and label each node with the order in which it is expanded. Also show the current game board at each node in the tree. Once again, use the graph search version of the algorithm.

Initial State

1		3
4	2	5
7	8	6

Goal State

1	2	3
4	5	6
7	8	

4. *[10 points]* In the above problem, we used the Manhattan distance of all tiles from their goal positions as a heuristic for the 8-puzzle (referred to in the book as h_2). We were very specific that the blank does not count as a tile. Consider a heuristic h_{2b} that includes the Manhattan distance of the blank from its goal position (in addition to those of the tiles). Is this heuristic better or worse than h_2? Why?
5. *[40 points]* This problem looks at playing the game tic-tac-toe. Assume that X is the MAX player. Let the utility of a win for X be 10, a loss for X be -10, and a draw be 0. There are two parts to this question, each using one of the two game boards given below:

board1

	O	
X	O	X
	X	O

board2

		X
	X	O
O		

   a) Given the game board **board1** above where it is X's turn to play next, show the entire game tree. Mark the utilities of each terminal state and use the minimax algorithm to calculate the optimal move.

b) Given the game board **board2** on the opposite page where it is X's turn to play next, show the game tree with a cut-off depth of two ply (i.e., stop after each player makes one move). Use the following evaluation function on all leaf nodes:

$$Eval(s) = 10X_3(s) + 3X_2(s) + X_1(s) - (10O_3(s) + 3O_2(s) + O_1(s))$$

where we define $X_n(s)$ as the number of rows, columns, and diagonals in state *s* with exactly *n* X's and no O's, and similarly define $O_n(s)$ as the number of rows, columns, and diagonals in state *s* with exactly *n* O's and no X's. Use the minimax algorithm to determine X's best move.

Extra Credit *(+20 points)***:**
This optional exercise requires you to do some Java programming in order to conduct an experiment that compares uniform-cost, greedy best-first, and A* search. In order to do this, download and read the code I have made available from the course web site (under Additional Class Materials). This code implements all three search algorithms and provides two *abstract* classes: SearchProblem and State. In order to solve a particular problem, you only need to extend these two classes with details specific to your problem.

Your task is to extend the code to solve the following three configurations of the 8-puzzle, where the goal state for all three configurations is as specified to the far right.

Initial State #1

7	1	4
6	3	2
	8	5

Initial State #2

4	8	2
6	3	5
1		7

Initial State #3

7	5	3
6		4
8	1	2

Goal State

1	2	3
8		4
7	6	5

You will then run each of the three search algorithms on each puzzle, recording the path cost of the solution (if found), number of nodes expanded, number of nodes generated, and time to perform the search. To do this, you'll need to write a class that extends State and can record the state of the game (i.e., the current position of each tile). It is important that this class implements an equals() method that can be used to compare the current state to another state. You'll need to write a second class that extends SearchProblem and implements the four methods: getInitialState(), goalTest(), getSuccessors(), and getHeuristicValue(). You may want to include a constructor that allows you to initialize the class with different initial states. Note, getSuccessors() returns a list of Successor elements, where each Successor records the State, a string describing the action to reach it, and the step cost of executing that action. By including the step cost in the successor information, we can avoid providing a separate path cost function. For the heuristic, use the sum of the Manhattan distances of all tiles from their goal positions. Finally, you'll need to write a main() method that runs the tests.

After you collect your data, write an analysis of it. What appears to be the strengths and weaknesses of each algorithm based on your experiment? Did the experimental results agree with the theoretical properties of the algorithms discussed in class and in the book? What, if anything, surprised you?

Attach a hardcopy of your code, the output of your experiment, and your analysis to your homework submission. Submit your source (.java) and compiled (.class) files to via e-mail to heflin@cse.lehigh.edu with subject line: "CSE 327: HW #2 Extra Credit".