EECS 391 Introduction to Artificial Intelligence

Programming Assignment 1

In this assignment, you will explore problem solving by search by writing AI algorithms to solve the 8-puzzle. A description and discussions of this problem are in Chapter 3 of the textbook. The puzzle begins with scrambled tiles, and the goal is to move the tiles (or equivalently the blank "tile") so that the blank tile in the upper left corner and the numbers are in order as shown.

You can do the assignment in either Java, python, or julia. If you would like to use a different language, you need to check with me (Prof. Lewicki) within a week of when the assignment is handed out. The only constraint that we have to be able to run and test your assignment at least one of our computers using standard installations and libraries.

Your code will read text input for commands and write the solution as text output.

Commands

Your code should read sequences of commands from a text file that operate on the current puzzle state. The text file should specified as an argument to your program. Output should be printed to the terminal (stdout). You will implement the following commands:

`setState <state>`
Set the puzzle state. The argument specifies the puzzle tile positions with a sequence of three groups of three digits with the blank tile represented by the letter 'b'. For example, '1b5' specifies a row with 1 in the left tile, nothing in the middle tile and 5 in the right tile. The goal state is "b12 345 678".

`printState`
Print the current puzzle state.

`move <direction>`
Move the blank tile 'up', 'down', 'left', or 'right'.

`randomizeState <n>`
Make n random moves from the goal state. Note that the goal state is not reachable from all puzzle states, so this method of randomizing the puzzle ensures that a solution exists.

`solve A-star <heuristic>`
Solve the puzzle from its current state using A-star search using heuristic equal to "h1" or "h2" (see section 3.6, p. 102). Briefly, h1 is the number of misplaced tiles; h2 is the sum of the distances of the tiles from their goal positions. You are free to try other heuristics, but be sure

that they are admissible and describe them in your writeup. When the goal is found, your code should print the number of tile moves needed to obtain the solution followed by the solution as a sequences of moves (up, down, left, or right) from the starting state to the goal state.

`solve beam <k>`
Solve the puzzle from its current state by adapting local beam search with k states. You will need to define an evaluation function which you should describe in your writeup. It should have a minimum of zero at the goal state. When the goal is found, print the number of tile moves and solution as for A-star search.

`maxNodes <n>`
Specifies the maximum number of nodes to be considered during a search. If this limit is exceeded during search an error message should be printed.

Code and Design

You are encouraged to library functions for data structures such as arrays, lists, or hash tables. You should, of course, implement the search algorithms. You are free to define additional methods as you see fit.

Writeup

Your writeup should be tutorial in natural. Write as if you are sitting down with the grader and demonstrating to them your understanding of the problems and the correctness of the code.

1. Code Design

The first part of your write should describe how your code is organized and what design choices you made. You should include code in your writeup, but do not include not all of it. Only use the relevant portions for specific points.

2. Code Correctness

The next part of the writeup should demonstrate your code on examples with the goal of proving to the grader that your search algorithms function correctly. This should explain and illustrate how the search algorithms work and how they can fail. Choose examples that are concise and easy to verify.

These examples should be created by reading commands from a file, which you should include with your submission. Be sure to set the seed of your random number generator so that your code generates the same random examples across multiple runs. When we run your code, we should get the same output that you did.

Not everything you include in your writeup needs to have been generated from the command file. For example, you will need to write methods for the experiments below. The purpose of the command file is to allow the grader to check basic functionality and correctness.

3. Experiments

The next part of your writeup should include a set of experiments to answer the following questions that compare the different search methods. For these answers, you will need to collect statistics from different random initial stats as was done on p.104.

a. How does fraction of solvable puzzles from random initial states vary with the maxNodes limit?

b. For A* search, which heuristic is better?

c. How does the solution length vary across the three search methods?

d. For each of the three search methods, what fraction of your generated problems were solvable?

4. Discussion

The final part of your write up should discuss the following:

a. Based on your experiments, which search algorithm is better suited for this problem? Which finds shorter solutions? Which algorithm seems superior in terms of time and space?

b. Discuss any other observations you made, such as the difficulty of implementing and testing each of these algorithms.

Extra Credit

Re-use your search code to solve a 2x2(x2) Rubik's cube (or some other relatively simple puzzle). You can write a different set of commands for each puzzle (or a different program with the same commands), but you can should re-use the same code for your search algorithms for the 8 puzzle. For the 2x2 puzzle, use standard notation and colors. (Note: Part of the extra credit is to fill in the details yourself, so you don't need to ask me what I want. Just do something reasonable and explain it clearly.)

What to submit

Prepare a ZIP file that contains: 1) source code (do not include the binaries), 2) your writeup, which should be a PDF document, and 3) the command file(s) used for your writeup. Name your file as "yourname_P1.zip" and submit it via canvas.

Grading

Code that we cannot compile, run, or test will receive a maximum of 50% of the total grade. Please comment your code so we can more easily understand it, and use sensible variable names.

	%		Description & Criteria
commands	10%		Everything but the "solve" methods; graded on correctness
A* search	30%		Search algorithms are graded on correctness and design, which is the choice of data structures, algorithms, and code organization.
beam search	30%		
writeup	30%		The writeup is graded on the clarity of the explanations, choice of examples, and experimental parameters. Conciseness is encouraged, redundancy and unnecessary examples without a clear purpose are penalized.
1. code design		5%	
2. code correctness		10%	
3. experiments		10%	
4. discussion		5%	
total	100%		
extra credit	20%		(Bonus) The extra credit portion of the assignment does not have strict guidelines, but your write up should follow the format of the main assignment. Points will be award based on correctness, your description, and to what extent you were able to re-use your search code.