Exercise #1-Part 2:

Solving N-Puzzle Problem Using A\*

Attached are the following files:

1. frontier.py – Implementing frontier with a priority queue. Will be used to prioritize states based on a value function, state.hdistance(s)+state.path\_len(s). You will need to write two versions of state.hdistance(s) ☺
2. state.py – Implements an N-Puzzle, except for the heuristics. Your assignment!
3. search.py – Implements the search pseudocode from class.

You need to go through the attached files, understand them, and perform the following tasks:

1. Run the code “as-is” and note that this is uniform search (why?). Run the code 100 times for the 2X2, 3X3, and 4X4 puzzles. Save/print the output of the number of states inserted, checked (removed), solution depth and time as averaged from 100 (still not always possible for the 4X4 puzzle). What do you see? For purposes of completeness, I know that some of these searches will timeout. To get around this, I suggest that any time a search takes more than 5 seconds (e.g. check: 1000\*(toc-tic)), you give up and you add large numbers for the values to the counters (e.g. Average depth = 20, Average number inserted / removed = 10000, runtime=100 seconds).
2. Implement the hdistance1 function to accurately return the number of tiles out of place and change the code in frontier to return state.hdistance1(s)+state.path\_len(s). Run the code 100 times and and save/print the output of the number of states inserted, checked (removed), solution depth and time as averaged from 100 runs for the 2X2, 3X3, and 4X4 puzzles (should now be almost always possible in all cases).
3. Implement the hdistance2 function to accurately return the Manhattan distance of the tiles from their target position and change the code in frontier to return state.hdistance2(s)+state.path\_len(s). Run the code 100 times and save the output of the number of states checked (total items inserted and deleted) as averaged from 100 runs in a 2X2, 3X3, and 4X4 puzzles.
4. Compare these values to those you achieved in the first part and print a summary table that looks like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Depth | Inserts | Removes | Time |
| BFS |  |  |  |  |
| DFS |  |  |  |  |
| Uniform |  |  |  |  |
| A\*-h1 |  |  |  |  |
| A\*-h2 |  |  |  |  |

\* Reminder from the lecture:

