Jeremy Grifski

CSE 5521

September 12, 2019

Homework 3

1. Implement a successor function and goal test function for the 8-puzzle problem, as described in the slides. Refer to the provided template file eight\_puzzle\_student.js for more detailed instructions. (Also see the two\_jugs.js and vaccuum.js files for examples of these functions for some other problems.) (4 pts)

**See Attached**

1. Come up with several board configurations and use eight\_puzzle.htm to test your functions from (1) against them. (Be sure to include the goal state!) Do the results from your functions match your expectations? Explain. (1 pt)

**After playing with a few configurations, I can confirm that the goal test works well. Since I check the current state against a hardcoded goal state, I wouldn’t expect it to fail. As for the successor state function, it’s a bit harder to prove. To implement it, I used a mapping of action IDs to row, column vectors which represent the move in terms of integers. After some tweaking, I’m confident this method works as well. See tests below.**

|  |  |  |
| --- | --- | --- |
| 8 | 1 | 3 |
| 2 | 4 | 5 |
| 7 | 6 |  |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 |  | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 8 | 1 | 3 |
| 2 | 4 | 5 |
| 7 | 6 |  |

1. Implement the breadth-first search algorithm. Refer to the provided template file bfs.js for more detailed instructions. (6 pts)

Your search functions must be generic (i.e., they don’t depend on the problem you are solving). You should be able to use the provided example problems in two\_jugs.htm and vaccuum.htm as additional tests for your code.

**See Attached.**

1. Implement the depth-limited search algorithm. Refer to the provided template file dls.js for more detailed instructions. (6 pts)

**See Attached.**

1. Implement the iterative-deepening search algorithm. Refer to the provided template file ids.js for more detailed instructions. (2 pts)

**See Attached.**

1. Implement the A\* search algorithm. Refer to the provided template file astar.js for more detailed instructions. (4 pts)

**See Attached.**

1. Come up with several board configurations and test your 4 search functions on them (you may re-use the boards from (2) ). Run your depth-limited search twice, first using as the depth limit the length of the path returned by either your BFS or IDS. Second, use twice that value. Do the returned solutions (or lack thereof) match your expectations? Explain. (1 pt)

**First, this problem helped me isolate a bug in my DLS solution (off by one) and another bug in my BFS solution (using shift/unshift improperly). Then, I ended up with a bizarre behavior where the proper depth works but doubling the depth causes the path length to be larger. Of course, that makes sense because DLS is going to continually deepen until it hits the limit or the goal. That goal path is not necessarily the shortest path. When DLS is restricted to the proper length, it’s going to get it right.**

1. Choose a non-trivial board configuration and report the number of states evaluated and expanded for each search function. Run depth-limited search with two different depth values as in (7). Test A\* search using both the Misplaced Tile Count and Manhattan Distance heuristics (both have been provided for you in eight\_puzzle\_student.js). Also, test A\* using a “stupid” heuristic that returns only 0. Do these values match your expectations? Discuss. (2 pts)
2. On Thursday, I started working on this assignment right when I got home at 6PM and worked on it until about midnight—so ~6 hours. Of course, 4 of these hours was spent coding. The remaining time was spent completing the report.