Lab 1: Search

Published | Edit | ⋮

This assignment requires coding in Javascript. Use the template files provided in Lab1_template.zip (https://osu.instructure.com/courses/102347/files/32373882/download?download_frd=1) to get started.

---

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)

---

2) 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)

---

3) 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). Do not assume anything about the structure of the states you are given (e.g., do not assume they are objects). Make sure to test that the two example problems, *two_jugs.htm* and *vaccuum.htm*, also work with your code!

Note: See SearchImplNotes.pdf (https://osu.instructure.com/courses/102347/files/32373884/download?download_frd=1) for some additional discussion on some of the practical issues implementing search algorithms.

---

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

---

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

---

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

---

7) 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)

---

8) 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)

Create a report including answers to the asked questions (please use doc or pdf format, alternately txt format if you absolutely cannot do the other two).

Then, create a ZIP archive of your code files and report and submit it to the Lab 1 dropbox on Carmen.

(Please include the entire template in the zip, our grader should be able to download and unzip your submission and immediately run it, without copying anything in.)

Tips:

A large number of hints have been provided as comments in the code, please read them carefully.

If you need to print out debug statements, you may use the console.log() function to print out to the browser’s debug console. To access this log, use Ctrl-Shift-J in Chrome or Shift-F5 (Console tab) in Firefox.
Safari Option-Cmd-C

Alternately, you can use the helper_log_write() function (from search_helper.js) to output to the log region on the web page.

An example search function has been provided in rnds.js which shows how the is_goal_state() and find_successors() functions should be called and how the solution path returned by search functions should be formatted.