Slider Lab 1 – AI

17 Sep 2020

In class we discussed the structure of BFS (breadth first search) and DFS (depth first search) and saw pseudocode for solving an arbitrary slider puzzle.

Slider Lab 1: You should apply BFS to optimally solve an arbitrary *v* x *h* slider puzzle.

Implement a Python script whose first argument indicates the starting position of a slider puzzle. The second argument, if provided, is the goal state. The default goal state is `"12345678_"` for standard 3 x 3 slider puzzles, where the _ indicates an empty position. More generally, the default goal state will be the non-underscore characters of the starting slider, in sorted order, followed by the underscore. The dimensions of the slider puzzle are taken to make it as "square" as possible. Specifically, $v * h$ should give the length of the input puzzle (as a string), and h should be the smallest integer such that $h \geq v$ (puzzles will be at least as wide as they are tall).

Your script should have a function `solve(puzzle, goal)` that implements the BFS pseudo-code shown in class. One of the helper routines you write should be `neighbors(puzzle)`, which returns those puzzles that are a distance of 1 away from `puzzle`. The distance between two puzzles is the minimum number of steps needed to get from one puzzle to the other. Thus, the return value of `neighbors(…)` should have two, three, or four items.

You can expect to be modifying the above two routines in the future, so please ensure that you have implemented what is stated above, the former having been shown in class. Because the dimensions of the slider puzzle won't change once they are established, you may treat them as globals (eg. `gWIDTH`, `gHEIGHT`).

Your script should output:

1) A sequence of states starting from the input `puzzle` up through the `goal` puzzle that constitute a minimum length path to the `goal` from `puzzle`. If the two are the same or if there is no solution, then only print out the starting `puzzle`. The number of steps needed in the former case is 0 (it takes 0 steps for a puzzle to get to itself).

Puzzles are to be printed out as 2D arrangements rather than a simple string. You should print anywhere from 5 to 12 puzzles going across (we will call a horizontal sequence of puzzles a band), but the final band of puzzles may have fewer than your selected number of puzzles going across. For example, if it takes 19 steps to get to your goal and you have selected 7 puzzles in a band, then you would show the first 7 puzzles, then the next 7, and for the final band show 6.

As a second example, if your script is called as `eights.py 1234567_8 12345678_` then you would have one band (horizontal grouping) of two puzzles in the output.

2) Your script should output the test `Steps: #` where # is the number of steps in the minimal solution. If there is no solution, then output `Steps: -1`.

3) As the final line of your output, you should print out `Time: #s` where the # is the number of seconds (should have three significant digits).