Summary:

My problems this week were, first and foremost, being really busy early in the week, and when I wasn’t not, at all, being in the problem comprehending (much less solving) mood. This kept me from actually starting work on the code until Thursday. Once I did, it didn’t take much to get it running, but once it was, I found that it can’t solve puzzles that require it to take a lot of steps. Looking at its output, it looks like it gets stuck in a loop.

I ended up solving it, which I described in my emails to you, so I’m just going to leave it there.

1. Compare the number of board states being searched for 2x2, 3x3 and 4x4 boards. Vary the number of shuffles. What do these numbers tell you about the complexity of the higher level puzzles?
   1. The larger boards tend to take more steps because there’s a larger number of possible moves each turn. For instance, on a 4x4 that I shuffled 19 times, the total nodes looked at was something like, 8k (I stopped it early because it had been running for about 5 minutes and had only reached depth 10).
2. Add a cout << or display the depth level in your algorithm whenever the depth increments. Try various amounts of shuffles from a few to a few hundred. Can your IDFS solution find a solution quickly on all 3x3 boards? 4x4 boards? What about 5x5? 6x6? Explain in terms of how much depth your algorithm is searching for each of these types of boards.
   1. As mentioned above, the more depth, the longer it takes. Generally, if it needs to search above a depth of 10, it starts taking multiples minutes to solve.
3. Are there any other interesting experiments you were able to run using your IDFS algorithm?
   1. I’d imagine it would be cool to use it to try to solve a Sudoku board, or to figure out a password.
4. Give your h[3] heuristic function that you developed in Part 2 of the lab. Explain why you think this is a good heuristic and how you developed it.
   1. h[3] = the number of times the board was shuffled. If you keep track of how many times the board was shuffled, like I do in my code, than you know that it should only take that number of moves to get it unshuffled. I thought of it because it’s what I’m currently using to figure the maxDepth in my program.