Isaac Ng AI Extra Credit

1. My code is fairly inefficient but very complete. I have the two requisite classes BeamSolver and StarSolver along with the Rubik's cube representation class RubikTwo and the Reader class, which executes all the methods and reads in all the commands. I represented my Rubik's cube using a character array of length 24 and as a char[8][3] array. The char[8][3] made the heuristics easier to calculate and were the only reason they were there. The char array of length 24 was super simple and easy to loop through and manipulate throughout the code. In both my starSolver and beamSolver, I used Priority Queues and Hashsets. The Priority Queue allowed me to sort the children and the next moves by the best move available judging by the heuristics. The hashsets enabled me to check if I had seen the state before and eliminate that state if need be. Reader uses simple bufferedReaders and command line reading to do the requisite methods.
   HashSets:
   Set<boardNode> kbest = new HashSet<boardNode>();
   Set<RubikTwo> seen = new HashSet<RubikTwo>();
   PriorityQueue which utilizes comparator of type heuristic:
   PriorityOrder order = new PriorityOrder(heuristic);
   PriorityQueue<boardNode> queue = new PriorityQueue<boardNode>(order);
   Differing representations:
   public char[] cube = new char[24];
   char[][] cubearray = new char[8][3];
2. A star will always find a solution if there is one and if there isn't one, it will report that. We can adjust how far A star will search for it by setting the maxNodes, so if you set it to 1, then A star can't find the solution because the search will be cut off. Beam Search sometimes never converges if there is a high randomize state integer due to never converging and getting caught in a loop somewhere. 10,000 is a reasonable number as A star takes around 2000 to find the solution while beam search will converge around 10000. The file can be run using "java Reader input.txt". My two searches both output the solution moves and the move count if available and if with a short randomizeState integer.
3. Experiments
   a. As the maxnodes increases, the precentage of solvable states increases. Around 10000, the proportion levels off, but going from 1000 maxnodes (5% solvable) to 10000 maxnodes (99% solvable) shows that the maxnodes limit increases the fraction of solvable puzzles.
   b. H2 is better than h1. If there are n misplaced tiles, h1 will give n, while h2 will always give some value that is bigger than n, lending itself to be a better heuristic.
   c. A star will always find optimal solutions, while beam search finds slightly not optimal solutions. Beam search either converges to the solution or can't find a solution after exploring many nodes.
   d. A* finds solutions if they exist and the maxNodes is big enough. That means that it will solve 100% of the solvable states. Beam Search solves about 99% of the states with a beam width of around 300. With a width of 100, around 95% are solvable. With a width of 50, around 70% and with a width of 5, around 5%.
4. Discussion

a. A star search using h2 is the best way to solve the problem. H2 finds the shorter solutions as h1 has to explore many more nodes to find the solution, if it can even find one. Beam search is better when it comes to space, but A star is both optimal and complete and better with time complexity.
b. Beam search for me was a nightmare to implement as I couldn't quite figure out how to generate the k best states and cycle through them to find the next k best states. A star was relatively simple as I had implemented it correctly in my previous code. A star also tended to always find a solution and take less time, so it was easier to test and gather data from while beam search tended to take a long time with no guarantee of a solution.