Report

Iterative Deepening Search (IDS):

The IDS that I implemented created a list of lists, initialized with the starting input. For each element that was in the beginning list, it produces the children nodes and checks if they are the solution. If they are the solution, the program ends and the result is returned. However, if it is not, then once all elements of the first list are explored, then it is removed, and the cycle continues until an answer is either found or the list is empty. This is done so that all items at a particular depth are explored before moving onto the next depth.

An example of the conceptualness of this would be to have the node 1 be at depth 0, nodes 2,3,4 be at depth 1, 5, and 6 as the children of 2, 7 as the child of 4. This list discussed would follow the following steps for execution:
[[]] ⇒ [[1]] ⇒ [[1],[2,3,4]] ⇒ [[2,3,4]] ⇒ [[2,3,4],[5,6]] ⇒ [[2,3,4],[5,6],[7]] ⇒ [[5,6],[7]] ⇒ [[7]] ⇒ [[]]

A* Search:

For implementation of this search algorithm, I used 3 different heuristics. I used Manhattan Distance, Number of Moveable Pegs, and Distance From Center. The way the algorithm works is by creating a queue of nodes to explore, initialized with the starting input. It goes through the list and finds the node with the least cost (calculated by summing the depth and the heuristic value). Once the node is found, it checks if the node itself is a solution, if it is, it outputs the necessary information. If it is not, on the other hand, then it appends its value to a list of explored inputs and proceeds to produce the children. Once the child nodes are created, it then checks if they already exist in the list of explored nodes. If the child node has not been explored, it adds a dictionary containing its cost and configuration to the queue. Once this has been done, it then removes the parent node from the queue. This continues until either an answer has been reached, or the queue is empty.

Experimentation Results:

For experimentation, I used as input (given in the Test_set_and_output document) the example configuration, a complicated configuration to which no answer existed, a complicated configuration to which an answer did in fact exist, a configuration where the input was the answer, and a configuration where the input was blank. The example configuration produced a time of about 8.12 seconds in the IDS, and an average time 3.74 for the A* search. From this typical search scenario, you can see that already the A* search has a better time. The IDS needed to go through 30 nodes before reaching a solution. While the A* search only needed to go through 18-24 nodes. In this scenario, the amount of memory usage between the IDS and the A* search were extremely similar (IDS memory was 19238912 bytes while A* was on average 19263488 bytes). In the situation where an answer did not exist, we see something contrary to the previous example. The result using the IDS takes about 3.29 seconds, while the A* search takes about 5.06 seconds with the quickest solution from the Manhattan Distance at 5.00 seconds, followed by the number of moveable pegs at 5.06 seconds. Given that the A* search has a list of already

explored nodes implemented, it is understandable that the number of explored nodes for A* is on average 796, while for IDS, it is 4948. The memory taken by the IDS was consequently bigger at 20631552 bytes, while for the A* search, it was on average 19394560 bytes. From the complicated input with a solution, we see that the IDS takes the most time at about 93 seconds, while Manhattan Distance takes about 3 seconds and Number of Moveable Pegs and Distance From Center is about 14 seconds. We also see that the amount of nodes visited with the IDS is 1191585, with the A* search and Manhattan Distance is 73, with the A* search and Number of Moveable Pegs is 13322, and with with the A* search and Distance From Center is 147. Given these findings, I believe that the most advantageous searches are A* with Manhattan Distance heuristic, followed by A* with Distance From Center heuristic, followed by A* with Moveable Pegs heuristic and IDS.

The other two scenarios tested were relatively the same amongst all searches and should not be taken into consideration because they did not actually go into the search algorithm.