# Assignment #1 Report

**Methodology**:

**Applies to all –** For parameters we have it setup to accept exactly what the assignment asks, a file with the initial state, a file with the goal state, the mode to use, and the name of the file to output it to. The first thing that we do with this is take the initial state file and the goal state file and retrieve the states from them, putting them each into a string. The string just has the numbers in order from left to right top to bottom of the puzzle. We figured that it would be easier and faster to deal with strings then 2D arrays. Then based on the mode that was inputted we call the corresponding function and send in the initial state and the goal state. Each algorithm has its own file that contains at least three functions. The first is a function named after the algorithm. This function creates a dictionary, because we're using python, for the closed list and an array for the fringe. Then it goes into a loop that takes nodes off of the fringe and expands them, checking to make sure that they're not in the closed dictionary, until either a solution is found or the fringe is empty which means that no solution exists. We chose to represent each state as a node object so that it could hold information like its state and parent. The second function in every algorithm file is expand which takes the current state and expands to find all its possible moves and then makes each one into a node and adds them to the fringe. The final function is swap which takes the string for the state and swaps two numbers to represent the movement that was made. The majority of this part came from the pseudo code from the book for Graph Search.

**Breadth-First Search** – The only thing extra that we really had to do for the BFS was make sure that for our fringe array we took stuff out of the array using FIFO. Originally we were also using an array for our list of closed nodes but we found that this was way too time consuming so we switched it to a dictionary which drastically increased the speed of our algorithm.

**Depth-First Search** – For DFS we were able to use everything from BFS. The only thing that we had to change was the way that nodes were taken out of the fringe so that it used LIFO instead of FIFO. The report wants us to say the depth limit that we used for DFS but the assignment never asked us to use one. We didn't set one up because the purpose of the depth limit is to get rid of infinite loops but since we're using a graph search we don't have to worry about that anyways.

**Iterative Deepening Depth-First Search** – For IDDFS we had to add a new value to the node object, depth. This keeps track of the number of moves from the initial state. We use this so that for each iteration of IDDFS it stops once it reaches a node of a certain depth.

**A-Star** – We created a new function in this file for the heuristic. We also added a new value to the node object to hold the f(n) value used in A-Star. We chose to use the h2 heuristic from the slides, the one that calculates the distance for each number to get to its goal location and then adds them all up. We chose this one because according the slides it was considered the better heuristic of the two.

**Testing** – How we tested our code was by having it run each of the test cases given for the assignment. If it returned the correct results, test1, or feasible results, test2 and test3, then we assumed that it worked on the test cases. We also created a few of our own simple cases and made sure that each algorithm tracked them as we expected them to.

**Results**:

Test 1

| Algorithm | Nodes to Solution | Nodes Expanded |
|---|---|---|
| BFS | 2 | 6 |
| DFS | 2 | 2 |
| IDDFS | 2 | 3 |
| ASTAR | 2 | 6 |

Test 2

| Algorithm | Nodes to Solution | Nodes Expanded |
|---|---|---|
| BFS | 4 | 26 |
| DFS | 32 | 32 |
| IDDFS | 4 | 59 |
| ASTAR | 4 | 26 |

Test 3

| Algorithm | Nodes to Solution | Nodes Expanded |
|---|---|---|
| BFS | 6 | 76 |
| DFS | 21058 | 181006 |
| IDDFS | 6 | 155 |
| ASTAR | 6 | 77 |

**Discussion**:

We did receive similar results to what I expected. The results for test 1 seem to all find the same solution path and expand close to the same number of nodes. Test 2 shows similar results as well. Except for test 2 the DFS seemed to have found a different solution then the rest. It seems that DFS is often just luck. If there happens to be a solution early on that initial path then it gets great results otherwise it could take a while. Test 3 is a good example of this. It appears that there was a solution after 6 moves but there was also one after 21058 moves which is the path that the DFS went over first so that is the one it returned. If we had chosen to use a limit for the DFS then this number probably could have been shortened a bit, maybe even to the point the point where it was comparable with IDDFS. It appears that for each test A* did about as well as BFS. This is because there was always a solution pretty early on which means that A* is about as good as BFS, depending on the heuristic used. If we had used a test case with more nodes to the solution then A* probably would have showed much better results than the rest.

**Conclusion**:

From these results I can conclude that for really short searches BFS is usually the best search to use, unless DFS gets lucky. Otherwise IDDFS works quite well and A* works even better, if it's using a good heuristic. One thing that these results show very nicely is which algorithms are optimal. Obviously DFS isn't since it had multiple instances of finding much worse solution paths. These results seem to match with what we expected to see based on information in the class sides and the textbook.