A* Search Algorithm to solve 8 Puzzle problem

The approach taken in solving the problem involved first implementing the two heuristics to measure how close a certain state is to the goal state. The first heuristic measures the total manhattan distance that each tile must travel to get to its goal state, while the second heuristic measures the number of misplaced tiles. The second function is a very broad measure of the 8 puzzle board, only taking into consideration the number of tiles that are in the wrong position. The first heuristic though takes into consideration the distance the misplaced tile is from its goal as well, providing a finer grained approach and giving a better measure of how close or far a state is from the goal. The next step was to implement the A* algorithm to solve the problem with the different heuristics.

The puzzle problem is represented as a graph of nodes containing the values and pointers to adjacent tiles. I felt this approach allowed for a more natural way of thinking of how the tiles move as well as implement it. I also implemented a way to create a board graph from a string of numbers and to get a string of numbers representing the current board state in order to be able to interact with the graph more efficiently.

Implementing the heuristics wasn't too difficult, the first consisted of iterating through the puzzle represented as an array to find any of the values that didn't match the key, incrementing the count for each one to represent the number of misplaced tiles. Implementing the second heuristic was a bit more involved, finding the vertical and horizontal distance the tile was from it's goal before adding it to a total count that represented the total manhattan distance for the state of the puzzle.

The A* algorithm was implemented through the use of a hash table and a priority queue, the table to record states that have been visited to prevent loops, and a queue to determine which of the states to explore next given the lowest current path cost plus heuristic cost. The algorithm continually polled the queue and added any states not yet seen to the queue with an updated path cost. The priority queue is sorted by both the current path cost associated with the board state as well as the heuristic cost calculated at runtime.

	Avg # of nodes explored		Avg # of µs taken	
Depth	Manhattan Dist	Misplaced Tile	Manhattan Dist	Misplaced Tile
2	6	6	213	225
4	9	9	155	131
6	14	16	124	148
8	20	30	111	159
10	32	67	231	515
12	61	159	474	1461
14	113	383	916	4151
16	212	865	1587	8665
18	365	2047	2201	18162
20	627	5033	3908	48725

The above table shows the average statistics generated from running 100 cases of each of the listed depths. Something interesting to note is that the time taken for both heuristics for the shallowest depths take longer for than those at the middle.