Fiona Baenziger

01/25/18

Assignment #1: Eight Puzzle

Video: https://youtu.be/VONsCBjIVPE

Output: * This output is different from in the video. I felt this was more efficient, so I reprogrammed*

The output is path from the root node to the solution. Includes Game State, Cost, Move & Overall Cost.

BFS – Easy

```
Game State: 134862705 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 134802765 | Cost: 6 | Move: UP     | Total Cost: 6
Game State: 134820765 | Cost: 2 | Move: RIGHT  | Total Cost: 8
Game State: 130824765 | Cost: 4 | Move: UP     | Total Cost: 12
Game State: 103824765 | Cost: 3 | Move: LEFT   | Total Cost: 15
Game State: 123804765 | Cost: 2 | Move: DOWN   | Total Cost: 17

Solved!
Game State: 123804765
Length: 5
Cost: 17
Time: 42
Space: 36
```

BFS – Medium

```
Game State: 281043765 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 081243765 | Cost: 2 | Move: UP     | Total Cost: 2
Game State: 801243765 | Cost: 8 | Move: RIGHT  | Total Cost: 10
Game State: 810243765 | Cost: 1 | Move: RIGHT  | Total Cost: 11
Game State: 813240765 | Cost: 3 | Move: DOWN   | Total Cost: 14
Game State: 813204765 | Cost: 4 | Move: LEFT   | Total Cost: 18
Game State: 813024765 | Cost: 2 | Move: LEFT   | Total Cost: 20
Game State: 013824765 | Cost: 8 | Move: UP     | Total Cost: 28
Game State: 103824765 | Cost: 1 | Move: RIGHT  | Total Cost: 29
Game State: 123804765 | Cost: 2 | Move: DOWN   | Total Cost: 31

Solved!
Game State: 123804765
Length: 9
Cost: 31
Time: 360
Space: 236
```

BFS – Hard

```
Game State: 567408321 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 507468321 | Cost: 6 | Move: UP     | Total Cost: 6
Game State: 057468321 | Cost: 5 | Move: LEFT   | Total Cost: 11
Game State: 457068321 | Cost: 4 | Move: DOWN   | Total Cost: 15
Game State: 457368021 | Cost: 3 | Move: DOWN   | Total Cost: 18
Game State: 457368201 | Cost: 2 | Move: RIGHT  | Total Cost: 20
Game State: 457368210 | Cost: 1 | Move: RIGHT  | Total Cost: 21
Game State: 457360218 | Cost: 8 | Move: UP     | Total Cost: 29
Game State: 450367218 | Cost: 7 | Move: UP     | Total Cost: 36
Game State: 405367218 | Cost: 5 | Move: LEFT   | Total Cost: 41
Game State: 045367218 | Cost: 4 | Move: LEFT   | Total Cost: 45
Game State: 345067218 | Cost: 3 | Move: DOWN   | Total Cost: 48
Game State: 345267018 | Cost: 2 | Move: DOWN   | Total Cost: 50
Game State: 345267108 | Cost: 1 | Move: RIGHT  | Total Cost: 51
Game State: 345267180 | Cost: 8 | Move: RIGHT  | Total Cost: 59
Game State: 345260187 | Cost: 7 | Move: UP     | Total Cost: 66
Game State: 340265187 | Cost: 5 | Move: UP     | Total Cost: 71
Game State: 304265187 | Cost: 4 | Move: LEFT   | Total Cost: 75
Game State: 034265187 | Cost: 3 | Move: LEFT   | Total Cost: 78
Game State: 234065187 | Cost: 2 | Move: DOWN   | Total Cost: 80
Game State: 234165087 | Cost: 1 | Move: DOWN   | Total Cost: 81
Game State: 234165807 | Cost: 8 | Move: RIGHT  | Total Cost: 89
Game State: 234165870 | Cost: 7 | Move: RIGHT  | Total Cost: 96
Game State: 234160875 | Cost: 5 | Move: UP     | Total Cost: 101
Game State: 230164875 | Cost: 4 | Move: UP     | Total Cost: 105
Game State: 203164875 | Cost: 3 | Move: LEFT   | Total Cost: 108
Game State: 023164875 | Cost: 2 | Move: LEFT   | Total Cost: 110
Game State: 123064875 | Cost: 1 | Move: DOWN   | Total Cost: 111
Game State: 123864075 | Cost: 8 | Move: DOWN   | Total Cost: 119
Game State: 123864705 | Cost: 7 | Move: RIGHT  | Total Cost: 126
Game State: 123804765 | Cost: 6 | Move: UP     | Total Cost: 132

Solved!
Game State: 123804765
Length: 30
Cost: 132
Time: 517721
Space: 73554
```

---

*A*2 – Easy*

```
Game State: 134862705 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 134802765 | Cost: 6 | Move: UP     | Total Cost: 6
Game State: 134820765 | Cost: 2 | Move: RIGHT  | Total Cost: 8
Game State: 130824765 | Cost: 4 | Move: UP     | Total Cost: 12
Game State: 103824765 | Cost: 3 | Move: LEFT   | Total Cost: 15
Game State: 123804765 | Cost: 2 | Move: DOWN   | Total Cost: 17

Solved!
Game State: 123804765
Length: 5
```

```
Cost: 17
Time: 11
Space: 11
```

---

*A*2 – Medium*

```
Game State: 281043765 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 081243765 | Cost: 2 | Move: UP     | Total Cost: 2
Game State: 801243765 | Cost: 8 | Move: RIGHT  | Total Cost: 10
Game State: 810243765 | Cost: 1 | Move: RIGHT  | Total Cost: 11
Game State: 813240765 | Cost: 3 | Move: DOWN   | Total Cost: 14
Game State: 813204765 | Cost: 4 | Move: LEFT   | Total Cost: 18
Game State: 813024765 | Cost: 2 | Move: LEFT   | Total Cost: 20
Game State: 013824765 | Cost: 8 | Move: UP     | Total Cost: 28
Game State: 103824765 | Cost: 1 | Move: RIGHT  | Total Cost: 29
Game State: 123804765 | Cost: 2 | Move: DOWN   | Total Cost: 31

Solved!
Game State: 123804765
Length: 9
Cost: 31
Time: 45
Space: 37
```

---

*A*2 – Hard*

```
Game State: 567408321 | Cost: 0 | Move: ROOT   | Total Cost: 0
Game State: 567480321 | Cost: 8 | Move: RIGHT  | Total Cost: 8
Game State: 567481320 | Cost: 1 | Move: DOWN   | Total Cost: 9
Game State: 567481302 | Cost: 2 | Move: LEFT   | Total Cost: 11
Game State: 567481032 | Cost: 3 | Move: LEFT   | Total Cost: 14
Game State: 567081432 | Cost: 4 | Move: UP     | Total Cost: 18
Game State: 067581432 | Cost: 5 | Move: UP     | Total Cost: 23
Game State: 607581432 | Cost: 6 | Move: RIGHT  | Total Cost: 29
Game State: 670581432 | Cost: 7 | Move: RIGHT  | Total Cost: 36
Game State: 671580432 | Cost: 1 | Move: DOWN   | Total Cost: 37
Game State: 671582430 | Cost: 2 | Move: DOWN   | Total Cost: 39
Game State: 671582403 | Cost: 3 | Move: LEFT   | Total Cost: 42
Game State: 671582043 | Cost: 4 | Move: LEFT   | Total Cost: 46
Game State: 671082543 | Cost: 5 | Move: UP     | Total Cost: 51
Game State: 071682543 | Cost: 6 | Move: UP     | Total Cost: 57
Game State: 701682543 | Cost: 7 | Move: RIGHT  | Total Cost: 64
Game State: 710682543 | Cost: 1 | Move: RIGHT  | Total Cost: 65
Game State: 712680543 | Cost: 2 | Move: DOWN   | Total Cost: 67
Game State: 712683540 | Cost: 3 | Move: DOWN   | Total Cost: 70
Game State: 712683504 | Cost: 4 | Move: LEFT   | Total Cost: 74
Game State: 712683054 | Cost: 5 | Move: LEFT   | Total Cost: 79
Game State: 712083654 | Cost: 6 | Move: UP     | Total Cost: 85
Game State: 012783654 | Cost: 7 | Move: UP     | Total Cost: 92
Game State: 102783654 | Cost: 1 | Move: RIGHT  | Total Cost: 93
Game State: 120783654 | Cost: 2 | Move: RIGHT  | Total Cost: 95
Game State: 123780654 | Cost: 3 | Move: DOWN   | Total Cost: 98
```

```
Game State: 123784650 | Cost: 4 | Move: DOWN   | Total Cost: 102
Game State: 123784605 | Cost: 5 | Move: LEFT   | Total Cost: 107
Game State: 123784065 | Cost: 6 | Move: LEFT   | Total Cost: 113
Game State: 123084765 | Cost: 7 | Move: UP     | Total Cost: 120
Game State: 123804765 | Cost: 8 | Move: RIGHT  | Total Cost: 128

Solved!
Game State: 123804765
Length: 30
Cost: 128
Time: 283480
Space: 48452
```

Analysis:

Does one solution seem better than the other? Explain. Identify failing solutions and why they failed?

After programming the algorithms and looking at the output, I think I may have programmed some of them incorrectly. When reviewing the tables, the BFS, Uniform Cost, Best First, A*1, and A*2 are all outputting very similar data about length, cost, time and space. Although there are some minute differences, such as time and space which ranged variably between these algorithms, I felt that they all performed similarly. The algorithm that stands out is Depth First Search – DFS – which performed horribly in comparison to the other algorithms. The depth is probably too great and is causing the search tree to grow too quickly and too large to justify it being an efficient solution to this problem.

Between the other algorithms, which I mentioned briefly already, they all have the same length & cost amongst each difficulty respectively. This is a red flag that I may have programmed it incorrectly as I don't believe that they call perform so similarly. Also, another red flag that something is incorrect is that nothing failed. There was never a timeout error or an overflow of the stack or so on. This does not make sense as I know some of these algorithms perform poorly with this specific eight puzzle specifications. Some differences between these seemingly similar algorithms (based on my designs) is regarding the time and space, which are also important factors when considering an optimal algorithm, but they are minute. Looking at the tables, A*2 with the Manhattan Distance heuristic performed the best.

I would like to justify my (interpretably) wrong output because I understand the algorithms, but I haven't programmed in a year and I spent this assignment reteaching myself algorithm design concepts. I would like to understand the correct style of programming algorithms such as these to get the correct outcome, so I am able to assess it correctly. I set up the program to be as concise as I could, creating a separate AINode class so I could customize what I want to see carried through the algorithm, like parent, children, path cost and more… I decided to program my game states as strings, so I have a function to interpret the children that are possible from a game state. Most of the algorithms I felt had the same basic structure but used different containers and assessed different values.

For Breadth First Search, which performed better than I had thought, I used a queue that I pushed the children to at the end of the solution and popped them off every iteration. The width as to what BFS goes to versus DFS appears to be more optimal for the Eight Puzzle as this performed more efficiently in all aspects.

For Depth First Search, it performed the worst out of all the algorithms. I could not even print out all the input as the solutions were so incredibly long. I think this is a fault of the algorithm as the depth of a problem like this is just too great to be efficient. I used a stack and popped at the beginning of every iteration and pushed the children at the end. I would not use this as a viable solution for this problem.

For Uniform Cost, this algorithm was relatively simple to program. I used a Priority Queue with a custom Comparator class to compare the value of G as we are prioritizing the nodes with a smaller distance from the root to N. All I did was update the G value of the children based on their parents G value and their path cost, which is what is getting compared for the Queue. The performance of this algorithm outputted the same Length and Cost as others but took more time and space.

The last three algorithms, they perform so statistically similar that I am not sure what I did to cause that to happen, but I know they assess different things which should thus affect their final path and cost. For Best First, it assesses a path using a heuristic that considers the number of misplaced tiles not in the correct position. This is a similar design to A*1, though A*1 assessed both g + h, also known as f, or the distance from the beginning to that point and the estimate of how much farther to go. I am not sure if I programmed these correctly because the output is almost the exact same despite Best First using time and space less efficiently.

For A*1 and A*2, I used the same design and in fact the same function, but I am computing a different heuristic which essentially should change the output but the meta output reveals that they perform almost identically to each other. This confuses me and I would like to know if it's a programming error or if they are just similar in that aspect.

Algorithm	**Length**	**Cost**	**Time**	**Space**
EASY				
BFS	5	17	42	36
DFS	109155	491193	120798	71050
Uniform Cost	5	17	24	16
Best First	5	17	13	9
*A*1*	5	17	14	11
*A*2*	5	17	11	11
MEDIUM				
BFS	9	31	360	236
DFS	106321	479501	115893	69554
Uniform Cost	9	31	135	90
Best First	9	31	67	52
*A*1*	9	31	68	54
*A*2*	9	31	45	37
HARD				
BFS	30	132	517721	73554
DFS	72686	326102	76102	50466
Uniform Cost	30	128	419251	53582
Best First	30	128	348044	48710
*A*1*	30	128	348045	48710
*A*2*	30	128	283480	48452