Jung Yeo

CS4200 AI

September 22, 2018

Prof Atanasio

Project 1

My Approach

This project made me use many different types of data structures to quickly and efficiently use the data that I created and quickly find the correct path. To emulate the tree type graph we went over in class I used a combination of a PQ and a hash map. The hash map stored all the visited nodes, with the board's ordering being the key. For example [0,1,2,3,4,5,6,7,8] = 12,345,678. I used this so I can quickly access data in O(1) time to check if visited node already exists. As for all the leaf nodes, I stored it in my PQ, since after insert of all possible combinations, I had to pull out the one with the least cost and use that to continue. Since a min PQ did that all on its own, I created a PQ with my own comparator class to accomplish what I wanted.

The nodes for each "possibility" were Board objects. Each object had an integer array, board, that held the game at that current iteration, a pointer to the parent that birth this child, and various other data points to use for calculations. The pointer to the parent was quite useful, since it allowed me, once I had my goal node, to trace back to the start node quickly.

The heuristics were basic math and counting problems. H1(incorrect placement) was just checking to see if a node was in the correct spot, if not I would add 1 to my returning value. Manhattan distance was a little more complicated, but after doing some research and converting a 3x3 grid into an array, I created a formula that would find the Manhattan distance for a single value. To find the total I just needed to calculated it for every node. (formula was = |value % 3 – index % 3| + | int(value\3) – int(index\3)|)

Compare and Analysis.

After computing the given test cases, as well as grabbing some random test cases, we can see that H2 is much more efficient than H1. H2 creates much less nodes, and takes a much shorter time. If we look at the values individually, near the lower end of the depth meter, both are roughly the same, but as the puzzle gets more complex, H2 shows how much more efficient it is. It appears as though H1 has a steeper exponential growth, H2, has a much more leveled increase over time.

Depth	Average Nodes	Total Test Cases	Avg Run Time
2	6	100	0
4	11	100	0
6	22	100	0
8	45	100	1

10	103	100	2
12	249	100	7
14	630	100	18
16	1564	100	53
18	3729	100	157
20	8654	100	503
21	14599	6	1085
22	23823	5	2429
23	36327	12	4926
24	52320	9	9401
25	75775	6	18180
26	112191	7	37993
27	154358	1	65977
28	225155	1	125903

Depth	Average Nodes	Total Test Cases	Avg Run Time
2	6	100	0
4	11	100	0
6	19	100	0
8	29	100	0
10	50	100	1
12	88	100	2
14	174	100	4
16	347	100	10
18	603	100	18
20	1004	100	31
22	2645	10	92
23	3784	23	150
24	3287	7	127
26	8399	4	448

Findings

H2 is by far the better heuristic when performing the 8tile puzzle when compared with the incorrect placement heuristic.