# CS363 AI Spring 2021

Student : Fengzhang Du

Problem Analysis:

| Method | Level | Optimal Path(Steps) | Nodes Expanded | Rum Time (ms) |
|---|---|---|---|---|
| A* Misplaced | Easy | 5 | 7 | 0 |
| | Medium | 9 | 163 | 4 |
| | Hard | 12 | 1007 | 4 |
| | Worst | java.lang.OutOfMemoryError: Java heap space | | |
| A* Manhattan | Easy | 5 | 6 | 1 |
| | Medium | 9 | 10 | 1 |
| | Hard | 12 | 904 | 10 |
| | Worst | java.lang.OutOfMemoryError: Java heap space | | |
| IDA* | Easy | 5 | 68 | 9 |
| | Medium | 9 | 911 | 13 |
| | Hard | 12 | 3395 | 20 |
| | Worst | 30 | 97103111 | 115663 |
| Depth First Branch and Bound | Easy | 5 | 35 | 0 |
| | Medium | 9 | 582725 | 882 |
| | Hard | 12 | 13 | 0 |
| | Worst | 30 | 40837279 | 48271 |

1. Number of possible states of the board: 9!
   Reachable states are 9!/2 = 181440
2. Average number of moves: 2*4+3*4+4 = 3
3. Sorry, no idea.
4. Sorry, no idea.
5. No. average branching factor is b = 3.
   Worst depth = 9!/2 = 181440 reachable states.
   Run time: b^d = roughly 181440^3. Not going to end before this semester.
6. Because the position of each tile is completely upside down and left to right, but the order is the same. For humans, we can solve it by pushing any tile of { 2,4,6,8 } to the middle and keep rotating the rest of the tiles on the edge until they are in the right position. Then push back the middle tile to its position.

But for Algorithms to solve it, it relies on the index/coordinates and heuristics. It will easily run into a tie for all the neighbor's h value and have to try almost all the possible moves in the end. So it's not very machine friendly.
Lessons for AI programs would be to make the algorithm think more like a human. Teach it the tricks or shortcuts, check edge cases before going to blindly solve it.

7. **A* : Assume h(n) is consistent.**

pros: Simply logic. Complete. Optimal. Tighter bound gives better solution.
Stops at first optimal solution.
cons: Runtime and Space complexity can be bad. Worst case for both: b^d.
Bad for large search problems.

**DFBnB :**

pros: Prune the node already worse than the current best solution.
When you find the first solution, you will find the optimal.
Use Linear Space.
cons: When m is infinite, it's not complete.
Need to set an upper limit to prevent the algorithm going too deep to a branch.
Use Exponential Run Time.
Not efficient for wider trees.

**IDA* :**

pros: Guarantee to stop and find the optimal solution.
Use Linear Space.
The Limits are expanded from lower to higher value. Opposite of DFBnB.
Use Linear Space.
cons: Use Exponential Run Time.
Not efficient for wider trees.

**Easy Optimal Sequence : The shortest path is 5 steps.**

Step : 0
1 3 4
8 6 2
7 0 5

Step : 1
1 3 4
8 0 2
7 6 5

Step : 2
1 3 4
8 2 0
7 6 5

Step : 3
1 3 0
8 2 4
7 6 5

Step : 4
1 0 3
8 2 4
7 6 5

Step : 5
1 2 3
8 0 4
7 6 5

**Medium Optimal Sequence : The shortest path is 9 steps.**

Step : 0
2 8 1
0 4 3
7 6 5

Step : 1
0 8 1
2 4 3
7 6 5

Step : 2
8 0 1
2 4 3
7 6 5

Step : 3
8 1 0
2 4 3
7 6 5

Step : 4
8 1 3
2 4 0
7 6 5

Step : 5
8 1 3
2 0 4
7 6 5

Step : 6
8 1 3
0 2 4
7 6 5

Step : 7
0 1 3
8 2 4
7 6 5

Step : 8
1 0 3
8 2 4
7 6 5

Step : 9
1 2 3
8 0 4
7 6 5

**Hard Optimal Sequence : The shortest path is 12 steps.**

Step : 0
2 8 1
4 6 3
0 7 5

Step : 1
2 8 1
4 6 3
7 0 5

Step : 2
2 8 1
4 0 3
7 6 5

Step : 3
2 8 1
0 4 3
7 6 5

Step : 4
0 8 1
2 4 3
7 6 5

Step : 5
8 0 1
2 4 3
7 6 5

Step : 6
8 1 0
2 4 3
7 6 5

Step : 7
8 1 3
2 4 0
7 6 5

Step : 8
8 1 3
2 0 4
7 6 5

Step : 9
8 1 3
0 2 4
7 6 5

Step : 10
0 1 3
8 2 4
7 6 5

Step : 11
1 0 3
8 2 4
7 6 5

Step : 12
1 2 3
8 0 4
7 6 5

**Worst: Optimal Sequence : The shortest path is 30 steps.**

Step : 0
5 6 7
4 0 8
3 2 1

Step : 1
5 6 7
4 2 8
3 0 1

Step : 2
5 6 7
4 2 8
0 3 1

Step : 3
5 6 7
0 2 8
4 3 1

Step : 4
0 6 7
5 2 8
4 3 1

Step : 5
6 0 7
5 2 8
4 3 1

Step : 6
6 7 0
5 2 8
4 3 1

Step : 7
6 7 8
5 2 0
4 3 1

Step : 8
6 7 8
5 2 1
4 3 0

Step : 9
6 7 8
5 2 1
4 0 3

Step : 10
6 7 8
5 2 1
0 4 3

Step : 11
6 7 8
0 2 1
5 4 3

Step : 12
0 7 8
6 2 1
5 4 3

Step : 13
7 0 8
6 2 1
5 4 3

Step : 14
7 8 0
6 2 1
5 4 3

Step : 15
7 8 1
6 2 0
5 4 3

Step : 16
7 8 1
6 2 3
5 4 0

Step : 17
7 8 1
6 2 3
5 0 4

Step : 18
7 8 1
6 2 3
0 5 4

Step : 19
7 8 1
0 2 3
6 5 4

Step : 20
0 8 1
7 2 3
6 5 4

Step : 21
8 0 1
7 2 3
6 5 4

Step : 22
8 1 0
7 2 3
6 5 4

Step : 23
8 1 3
7 2 0
6 5 4

Step : 24
8 1 3
7 2 4
6 5 0

Step : 25
8 1 3
7 2 4
6 0 5

Step : 26
8 1 3
7 2 4
0 6 5

Step : 27
8 1 3
0 2 4
7 6 5

Step : 28
0 1 3
8 2 4
7 6 5

Step : 29
1 0 3
8 2 4
7 6 5

Step : 30
1 2 3
8 0 4
7 6 5