CS3105 (Artificial Intelligence) Report Practical 1

Matriculation ID: 210021783

Overview

In this practical, we were tasked with implementing multiple different search algorithms to solve the sliding eights puzzle and showing our findings as to how efficiently they were at solving the puzzle.

In my implementation of this I almost fully implemented part 1, made an attempt at part 2 and unfortunately could not attempt part 3 due to time constraints and health issues.

Design & Implementation

Manhattan

I initially take the users input, which represents the initial puzzle state to be solved. I do this as a string because that's the safest way to take in user input before I start to validate the input. Although the specification did not say I should focus on input validation, I did spend quite some time focusing on it as I wanted the programs to be as robust as possible. I ensure that there are integers, all positive and within the range of nm – 1.

The puzzle's initial state is represented by a 2D array called initStateArray, which stores the numbers on the puzzle tiles. This is then used to create a board object which has the properties; puzzleState which represents the puzzle as a 2D int array, manhattanDistance which is the total manhattan distance of the puzzle, movesMade and moveToMake which are *supposed* to track the moves which unfortunately does not work as it should.

I then check if the puzzle is solvable in the first place by using the inversions count method that the specification tells us about. Then if it is already solved by checking if the manhattan distance is equal to 0.

I chose to use a priorityQueue to order the states to move into because I found it to be the most efficent way that I had prior experience with using.

In my main loop I have a hashset which keeps track of the states already visited, although the lectures did specifiy that BFS algorithms can have infinite loops, I found that without this, I would not be able to solve a puzzle that has anymore than 5 inversions. I would have made my code more modular by putting the main loop into it's own method but I did not have time to debug my code.

A*

It would have been more efficient for me to make this a subclass of manhattan since they are very similar, and to just override the method which contains the main while loop, but I was unable to get it fully working in time so just copied the whole program over and changed what had to be.

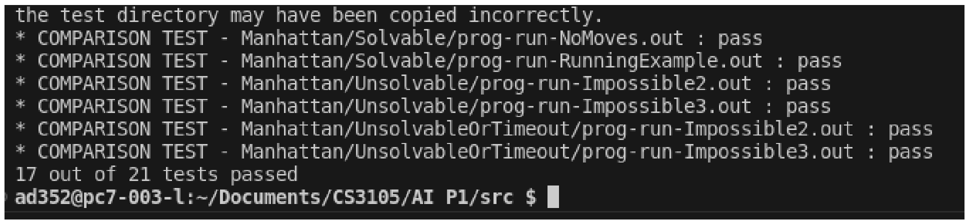
The main difference between my Manhattan implementation and my A* implementation is the way that the priority queue is organised, as I take into account the total moves made so far as well the the manhattan distance as the heuristic

Board

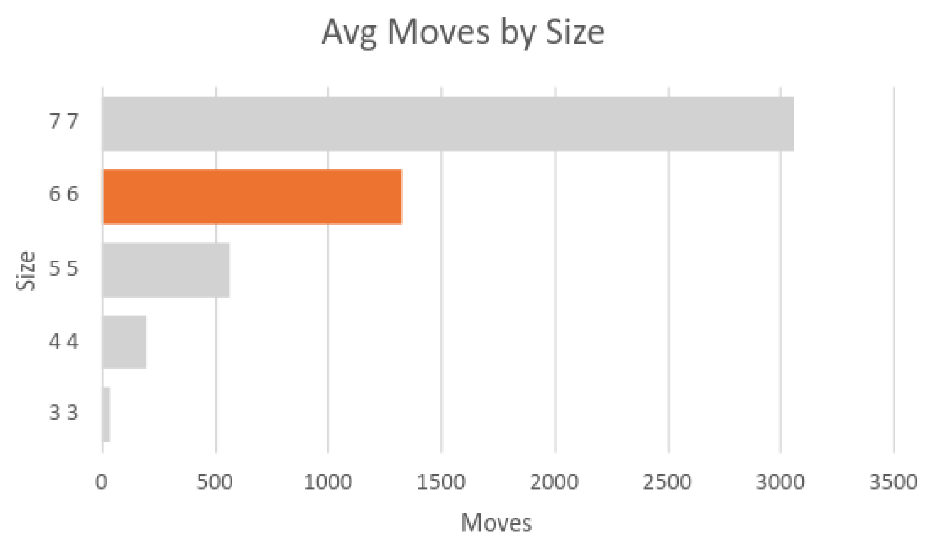
In this class, I keep everything related to the boards specifically. The main use of this is for the method getNeighbors(), I use this to retrieve the boards that are neighboring the empty space, what they would look like.

Testing

BASIC STACSCHECKS



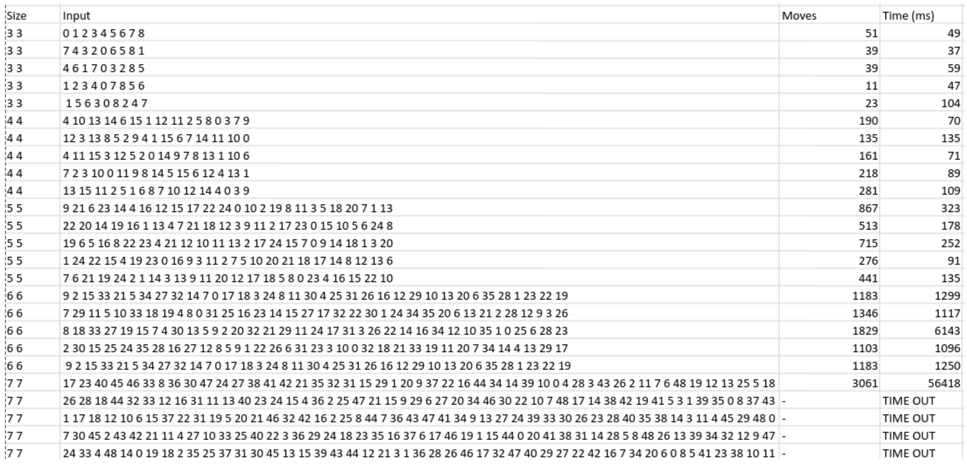
```
the test directory may have been copied incorrectly.
* COMPARISON TEST - Manhattan/Solvable/prog-run-NoMoves.out : pass
* COMPARISON TEST - Manhattan/Solvable/prog-run-RunningExample.out : pass
* COMPARISON TEST - Manhattan/Unsolvable/prog-run-Impossible2.out : pass
* COMPARISON TEST - Manhattan/Unsolvable/prog-run-Impossible3.out : pass
* COMPARISON TEST - Manhattan/UnsolvableOrTimeout/prog-run-Impossible2.out : pass
* COMPARISON TEST - Manhattan/UnsolvableOrTimeout/prog-run-Impossible3.out : pass
17 out of 21 tests passed
ad352@pc7-003-l:~/Documents/CS3105/AI P1/src $
```



A* Valid Tests

Size	Input	Moves	Time (ms)
3 3	0 1 2 3 4 5 6 7 8	21	97
3 3	7 4 3 2 0 6 5 8 1	19	72
3 3	4 6 1 7 0 3 2 8 5	19	35
3 3	1 2 3 4 0 7 8 5 6	11	51
3 3	1 5 6 3 0 8 2 4 7	17	56
4 4	4 10 13 14 6 15 1 12 11 2 5 8 0 3 7 9	-	TIME OUT
4 4	12 3 13 8 5 2 9 4 1 15 6 7 14 11 10 0	49	48641
4 4	4 11 15 3 12 5 2 0 14 9 7 8 13 1 10 6	-	TIME OUT
4 4	7 2 3 10 0 11 9 8 14 5 15 6 12 4 13 1	-	TIME OUT
4 4	13 15 11 2 5 1 6 8 7 10 12 14 4 0 3 9	-	TIME OUT
5 5	9 21 6 23 14 4 16 12 15 17 22 24 0 10 2 19 8 11 3 5 18 20 7 1 13	-	TIME OUT
5 5	22 20 14 19 16 1 13 4 7 21 18 12 3 9 11 2 17 23 0 15 10 5 6 24 8	-	TIME OUT
5 5	19 6 5 16 8 22 23 4 21 12 10 11 13 2 17 24 15 7 0 9 14 18 1 3 20	-	TIME OUT
5 5	1 24 22 15 4 19 23 0 16 9 3 11 2 7 5 10 20 21 18 17 14 8 12 13 6	-	TIME OUT
5 5	7 6 21 19 24 2 1 14 3 13 9 11 20 12 17 18 5 8 0 23 4 16 15 22 10	-	TIME OUT
6 6	9 2 15 33 21 5 34 27 32 14 7 0 17 18 3 24 8 11 30 4 25 31 26 16 12 29 10 13 20 6 35 28 1 23 22 19	-	TIME OUT
6 6	7 29 11 5 10 33 18 19 4 8 0 31 25 16 23 14 15 27 17 32 22 30 1 24 34 35 20 6 13 21 2 28 12 9 3 26	-	TIME OUT
6 6	8 18 33 27 19 15 7 4 30 13 5 9 2 20 32 21 29 11 24 17 31 3 26 22 14 16 34 12 10 35 1 0 25 6 28 23	-	TIME OUT
6 6	2 30 15 25 24 35 28 16 27 12 8 5 9 1 22 26 6 31 23 3 10 0 32 18 21 33 19 11 20 7 34 14 4 13 29 17	-	TIME OUT
6 6	9 2 15 33 21 5 34 27 32 14 7 0 17 18 3 24 8 11 30 4 25 31 26 16 12 29 10 13 20 6 35 28 1 23 22 19	-	TIME OUT
7 7	17 23 40 45 46 33 8 36 30 47 24 27 38 41 42 21 35 32 31 15 29 1 20 9 37 22 16 44 34 14 39 10 0 4 28 3 43 26 2 11 7 6 48 19 12 13 25 5 18	-	TIME OUT
7 7	26 28 18 44 32 33 12 16 31 11 13 40 23 24 15 4 36 2 25 47 21 15 9 29 6 27 20 34 46 30 22 10 7 48 17 14 38 42 19 41 5 3 1 39 35 0 8 37 43	-	TIME OUT
7 7	1 17 18 12 10 6 15 37 22 31 19 5 20 21 46 32 42 16 2 25 8 44 7 36 43 47 41 34 9 13 27 24 39 33 30 26 23 28 40 35 38 14 3 11 4 45 29 48 0	-	TIME OUT
7 7	7 30 45 2 43 42 21 11 4 27 10 33 25 40 22 3 36 29 24 18 23 35 16 37 6 17 46 19 1 15 44 0 20 41 38 31 14 28 5 8 48 26 13 39 34 32 12 9 47	-	TIME OUT
7 7	24 33 4 48 14 0 19 18 2 35 25 37 31 30 45 13 15 39 43 44 12 21 3 1 36 28 26 46 17 32 47 40 29 27 22 42 16 7 34 20 6 0 8 5 41 23 38 10 11	-	TIME OUT

Manhattan Valid Tests



Size	Input	Moves	Time (ms)
3 3	0 1 2 3 4 5 6 7 8	51	49
3 3	7 4 3 2 0 6 5 8 1	39	37
3 3	4 6 1 7 0 3 2 8 5	39	59
3 3	1 2 3 4 0 7 8 5 6	11	47
3 3	1 5 6 3 0 8 2 4 7	23	104
4 4	4 10 13 14 6 15 1 12 11 2 5 8 0 3 7 9	190	70
4 4	12 3 13 8 5 2 9 4 1 15 6 7 14 11 10 0	135	135
4 4	4 11 15 3 12 5 2 0 14 9 7 8 13 1 10 6	161	71
4 4	7 2 3 10 0 11 9 8 14 5 15 6 12 4 13 1	218	89
4 4	13 15 11 2 5 1 6 8 7 10 12 14 4 0 3 9	281	109
5 5	9 21 6 23 14 4 16 12 15 17 22 24 0 10 2 19 8 11 3 5 18 20 7 1 13	867	323
5 5	22 20 14 19 16 1 13 4 7 21 18 12 3 9 11 2 17 23 0 15 10 5 6 24 8	513	178
5 5	19 6 5 16 8 22 23 4 21 12 10 11 13 2 17 24 15 7 0 9 14 18 1 3 20	715	252
5 5	1 24 22 15 4 19 23 0 16 9 3 11 2 7 5 10 20 21 18 17 14 8 12 13 6	276	91
5 5	7 6 21 19 24 2 1 14 3 13 9 11 20 12 17 18 5 8 0 23 4 16 15 22 10	441	135
6 6	9 2 15 33 21 5 34 27 32 14 7 0 17 18 3 24 8 11 30 4 25 31 26 16 12 29 10 13 20 6 35 28 1 23 22 19	1183	1299
6 6	7 29 11 5 10 33 18 19 4 8 0 31 25 16 23 14 15 27 17 32 22 30 1 24 34 35 20 6 13 21 2 28 12 9 3 26	1346	1117
6 6	8 18 33 27 19 15 7 4 30 13 5 9 2 20 32 21 29 11 24 17 31 3 26 22 14 16 34 12 10 35 1 0 25 6 28 23	1829	6143
6 6	2 30 15 25 24 35 28 16 27 12 8 5 9 1 22 26 6 31 23 3 10 0 32 18 21 33 19 11 20 7 34 14 4 13 29 17	1103	1096
6 6	9 2 15 33 21 5 34 27 32 14 7 0 17 18 3 24 8 11 30 4 25 31 26 16 12 29 10 13 20 6 35 28 1 23 22 19	1183	1250
7 7	17 23 40 45 46 33 8 36 30 47 24 27 38 41 42 21 35 32 31 15 29 1 20 9 37 22 16 44 34 14 39 10 0 4 28 3 43 26 2 11 7 6 48 19 12 13 25 5 18	3061	56418
7 7	26 28 18 44 32 33 12 16 31 11 13 40 23 24 15 4 36 2 25 47 21 15 9 29 6 27 20 34 46 30 22 10 7 48 17 14 38 42 19 41 5 3 1 39 35 0 8 37 43	-	TIME OUT
7 7	1 17 18 12 10 6 15 37 22 31 19 5 20 21 46 32 42 16 2 25 8 44 7 36 43 47 41 34 9 13 27 24 39 33 30 26 23 28 40 35 38 14 3 11 4 45 29 48 0	-	TIME OUT
7 7	7 30 45 2 43 42 21 11 4 27 10 33 25 40 22 3 36 29 24 18 23 35 16 37 6 17 46 19 1 15 44 0 20 41 38 31 14 28 5 8 48 26 13 39 34 32 12 9 47	-	TIME OUT
7 7	24 33 4 48 14 0 19 18 2 35 25 37 31 30 45 13 15 39 43 44 12 21 3 1 36 28 26 46 17 32 47 40 29 27 22 42 16 7 34 20 6 0 8 5 41 23 38 10 11	-	TIME OUT

I conducted basic stacschecks and A* and Manhattan validity tests. Due to time constraints, I limited my testing of input validation, as it was not a primary focus of the specification. Through testing, I identified areas for improvement and refinement

PART 2 Questions

- As seen from the test data above, both Manhattan and A* were able to find optimal solutions for various problem sizes and instances. However, there are some instances where A* outperformed Manhattan in terms of the number of moves and execution time.

- Specifically, A* found optimal solutions with fewer moves and faster execution times in some 3x3 and 4x4 instances, which indicates that the chosen heuristic for A* might be more effective than the Manhattan heuristic in those cases.
- The programs were able to find solutions within 1 minute of CPU time for various problem sizes (3x3, 4x4, 5x5, 6x6, 7x7). However, there were instances (e.g., 4x4 and 7x7) where the execution time exceeded 1 minute (denoted as "TIME OUT").
- The size of the problem and the depth of the solution that can be reliably found within 1 minute CPU time may depend on factors such as the complexity of the puzzle, which I do not track but if I had more time I would have tried to find a way to, the quality of the heuristic, and the efficiency of the algorithm implementation.
- The efficiency of the program can be evaluated based on the execution times. In some cases, the execution times are reasonable, but in others, the program reaches a timeout condition, indicating that further optimization may be needed for solving larger or more complex

instances. To optimize the program, I would have considered implementing the heuristic differently.

Evaluation

In conclusion, my program is partially effective for its intended purpose, particularly with regards to solving the puzzle and displaying the move count. However, it falls short in correctly displaying the numbers moved into the empty space. Given more time, I would have dedicated additional effort to debug this issue. Nevertheless, my program can be deemed robust and reliable.

Conclusion

In summary, I am unsatisfied with my submission, considering the challenges I faced. Despite these constraints, I aimed to deliver a robust and functional solution.