Name: Julius Putra Tanu Setiaji Matric: A0149787E
Name: Tan Zong Xian Matric: A0167343A
Name: Tay Yu Jia Matric: A0171240W

1 Explanation

We implemented both programs using A* search but with two different heuristic functions: manhattan distance and linear conflict heuristics. We decided on using A* because it is the most efficient search algorithm to solve the 8-puzzle that we have learnt in the module so far. We use the manhattan distance heuristic because it is the one of the most common heuristics used for A* search to solve the 8-puzzle. We also decided to use the linear conflict heuristic because it is a refinement of the manhattan distance heuristic, thus we can compare the impact of heuristic accuracy on the performance of the A* algorithm

Definition 1 (Manhattan distance heuristic)**.** h_1 is the sum of the distances of the tiles from their goal positions. Since diagonal moves are not allowed, the distance is the sum of horizontal and vertical distances, also known as the manhattan distance.

The simplified algorithm for the linear conflict heuristic is as follows:

Definition 2 (Linear conflict)**.** Two tiles t_j and t_k are in a linear conflict if t_j and t_k are in the same line, the goal positions of t_j and t_k are both in that line, t_j is to the right of t_k, and the goal position of t_j is to the left of the goal position of t_k.

To derive the Linear Conflict estimate for any node n,

1. Calculate the minimum number of tiles that must be removed from row_1 such that there are no more linear conflicts.
2. Repeat Step 1 for the other rows and columns and sum them.
3. $LinearConflict(n) = 2\times$ result from Step 2

$$h_2(n) = ManhattanDistance(n) + LinearConflict(n) = h_1(n) + LinearConflict(n)$$

This heuristic works by augmenting the manhattan distance heuristic by adding the linear conflict estimate to it, thereby making it a more informed heuristic, tightening the lower bound. More detailed algorithm can be found in Hansson, Mayer, and Yung (1985, p. 13).

2 Statistics

Table 1: Statistics obtained from running the two programs on the given inputs

Heuristic	Input	No. of nodes generated	Max. size of frontier reached
h_1	`input_1.txt`	9	6
h_2	`input_1.txt`	9	6
h_1	`input_2.txt`	4250	1502
h_2	`input_2.txt`	2238	802
h_1	`input_3.txt`	181439	18722
h_2	`input_3.txt`	181439	19865
h_1	`custom.txt`	8824	2938
h_2	`custom.txt`	5934	2070

3 Analysis

3.1 Proof of Consistency of Heuristics in A* Search

Theorem 1. *Manhattan distance heuristic h_1 is consistent.*

Proof. For every node n, the step cost of getting to any of its successor node n', $c(n, n') = 1$ because for each move, only one tile is allowed to move. The tile that moves is either one step closer to or further away from its goal position, i.e. $h(n') = h(n) \pm 1$. Thus, $c(n, n') + h(n') = 1 + h(n) \pm 1 \geq h(n)$. Since $h(n) \leq c(n, n') + h(n')$, h_1 is consistent. □

Theorem 2. *Linear conflict heuristic h_2 is consistent (Hansson et al., 1985, p. 15).*

Proof. Let $md(n, x)$ be the manhattan distance of tile x in node n and $lc(n, r_i)$ be the number of tiles that must be removed from row r_i to resolve linear conflicts. Assume that tile x moves from row r_{old} to r_{new} while remaining on column c_{old}. Consider the effects of tile x's movement on $md(n', x)$ and $lc(n', x)$:

1. The goal position of x is in neither row. $md(n', x) = md(n, x) \pm 1$. There are no new linear conflicts. Hence, $h_2(n') = h_2(n) \pm 1$ and $h_2(n') + c(n, n') = h_2(n') + 1 \geq h_2(n)$.

2. The goal position of x is in r_{new}. Since x moved into its goal row, $md(n', x) = md(n, x) - 1$. This may or may not create new linear conflicts in row r_{new}, so $lc(n', r_{new}) = lc(n, r_{new})$ or $lc(n', r_{new}) = lc(n, r_{new}) + 2$. Because r_{old} is not the goal row of x, its absence would not contribute to any linear conflict in that row so $lc(n', r_{old}) = lc(n, r_{old})$. Hence, $h_2(n') = h_2(n) \pm 1$ and $h_2(n') + c(n, n') = h_2(n') + 1 \geq h_2(n)$.

3. The goal position of x is in r_{old}. Since x moved out of its goal row, $md(n', x) = md(n, x) + 1$. It may or may not have contributed to linear conflicts in r_{old} so $lc(n', r_{old}) = lc(n, r_{old})$ or $lc(n', r_{old}) = lc(n, r_{old}) - 2$. Because r_{new} is not the goal row of x, its presence would not contribute to any linear conflict in that row so $lc(n', r_{new}) = lc(n, r_{new})$. Hence, $h_2(n') = h_2(n) \pm 1$ and $h_2(n') + c(n, n') = h_2(n') + 1 \geq h_2(n)$.

In all three cases, h_2 remains consistent. By symmetry of the 8-puzzle, h_2 is similarly consistent for movements from column to column. □

3.2 Completeness and Optimality of Algorithms

Given that our heuristic functions are consistent, the graph-search version of A* search is *complete*, *optimal* and *optimally efficient* (Russell, Norvig, & Davis, 2010).

3.3 Comparison between A* Search Using the Two Heuristics

We created our own custom test case whose solution consists of 31 moves.

From table 1, the number of nodes generated by h_2 in the solveable cases is less than that generated by h_1. In the unsolveable cases, any search algorithm will expand all possible nodes before concluding that the puzzle is unsolveable as can be observed in table 1. From the lecture notes, the time complexity of A* Search is $O(b^{h^*(s_0)-h(s_0)})$. $LinearConflict(n) \geq 0 \implies h_2(n) = h_1(n) + LinearConflict(n) \geq h_1(n))$, thus h_2 dominates h_1 and $h^*(s_0) - h_2(s_0) \leq h^*(s_0) - h_1(s_0)$. Hence, A* Search with linear conflict heuristic has a lower time complexity than with manhattan distance heuristic. Indeed, in general, more dominant heuristics yield a more efficient A* search (Russell et al., 2010, p. 104).

From table 1, the memory consumption, measured by the max frontier size, is less than or equal to h_1. This is because the A* Search with h_2 reaches the goal node either as fast or faster than A* Search with h_1, thus A* Search with h_2 generates less than or equal to the number of nodes generated by A* search with h_1 along the way to the goal state. This is supported by results from table 1 as the relative proportions of memory consumption is similar to the proportions of number of nodes generated for the solvable test cases.

For the unsolvable test case, A* search using h_2 heuristic consumes more memory than A* search using h_1 heuristic. Results were similar for other unsolvable test cases of our own construction. We then ran the same unsolvable test cases using a third heuristic h_3, the number of misplaced tiles heuristic. h_3 showed higher memory consumption than both h_1 and h_2, which is consistent with the fact that both h_1 and h_2 dominates h_3. We are unable to fully explain these observations, but we have our own hypothesis.

In the unsolvable cases, the goal state is not one of the nodes that can be generated from expanding the initial state node. Thus, the heuristic functions never actually return 0, which breaks an assumption in the A* search optimality. Since h_2 has a tighter lower bound than h_1, the spread of values of $h_2(n)$ is narrower than those of $h_1(n)$. This results in many ties of nodes with high priority in the priority queue, which when expanded results in new nodes with similarly high priority, resulting in a larger max frontier size.

References

Hansson, O., Mayer, A. E., & Yung, M. (1985). Generating admissible heuristics by criticizing solutions to relaxed models.

Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial intelligence: A modern approach*. Prentice Hall.

Name: Julius Putra Tanu Setiaji Matric: A0149787E
Name: Tan Zong Xian Matric: A0167343A
Name: Tay Yu Jia Matric: A0171240W

Appendix A: custom.txt testcase

```
8 6 7
2 5 4
3 0 1
```