Final Project

15 Puzzle Solver

301441678

Dongan Kim

**Searching Algorithm:** Greedy search and A\*

**Heuristics:** Manhattan Distance + Linear Conflict:

**Linear Conflict**: Two tiles ‘a’ and ‘b’ are in linear conflict if they are in the same row or column, Linear Conflict combined with the Manhattan Distance to get the heuristics value of that state and each linear conflict will add 2 moves to the Manhattan Distance

h score = Manhattan distance + 2\* number of linear conflicts

**Ideas:** Solve by approaching 2tile by 2 tiles with Greedy search with Manhattan Distance + Linear Conflict as heuristics until it solves (total rows number -2) rows, once all (total rows number -2) rows get solved, then solves the rest of 2 rows as a whole with A\* algorithm.

First, try to solve 2 tiles by 2 tiles, 1 and 2, once 1 and 2 are placed to the right place, try to put 3 and 4 in right place and so on.

In order to solve 2tile by 2tile, make every number negative **except** the two target numbers (1 and 2).

**Start**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20 | 4 | 2 | 10 | 8 | 34 |
| 1 | 9 |  | 11 | 3 | 18 |
| 7 | 17 | 25 | 28 | 19 | 30 |
| 21 | 24 | 32 | 5 | 12 | 29 |
| 26 | 35 | 15 | 13 | 16 | 14 |
| 33 | 31 | 27 | 23 | 22 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -20 | -4 | **2** | -10 | -8 | 34 |
| **1** | -9 |  | -11 | -3 | -18 |
| -7 | -17 | -25 | -28 | -19 | -30 |
| -21 | -24 | -32 | -5 | -12 | -29 |
| -26 | -35 | -15 | -13 | -16 | -14 |
| -33 | -31 | -27 | -23 | -22 | -6 |

Modified Manhattan Distance and Linear Conflict to ignore negative number, so that it can focus on only the target numbers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 |  | -10 | -8 | 34 |
| -9 | -4 | -20 | -11 | -3 | -18 |
| -7 | -17 | -25 | -28 | -19 | -30 |
| -21 | -24 | -32 | -5 | -12 | -29 |
| -26 | -35 | -15 | -13 | -16 | -14 |
| -33 | -31 | -27 | -23 | -22 | -6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 |  | -10 | -8 | 34 |
| -9 | **4** | -20 | -11 | **3** | -18 |
| -7 | -17 | -25 | -28 | -19 | -30 |
| -21 | -24 | -32 | -5 | -12 | -29 |
| -26 | -35 | -15 | -13 | -16 | -14 |
| -33 | -31 | -27 | -23 | -22 | -6 |

Place 1 and 2 to right position change the target to 3 and 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -9 | -10 | -20 | -11 |  | -18 |
| -7 | -17 | -25 | -28 | -19 | -30 |
| -21 | -24 | -32 | -8 | -12 | -29 |
| -26 | -35 | -15 | -13 | -16 | -14 |
| -33 | -31 | -27 | -23 | -22 | -34 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| -9 | -10 | -20 | -11 |  | -18 |
| -7 | -17 | -25 | -28 | -19 | -30 |
| -21 | -24 | -32 | -8 | -12 | -29 |
| -26 | -35 | -15 | -13 | -16 | -14 |
| -33 | -31 | -27 | -23 | -22 | -34 |

New start board New goal board

Once the puzzle gets sorted the first row,

then set a new start board and goal board

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| -26 | -35 | -27 | -28 | -30 | -29 |
| -33 | -31 |  | -32 | -34 | -25 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 26 | 35 | 27 | 28 | 30 | 29 |
| 33 | 31 |  | 32 | 34 | 25 |

Once solve all (total rows number – 2) rows get solved, try to solve the rest of 2 rows as a whole with A\* algorithm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 26 | 35 | 27 | 28 | 30 | 29 |
| 33 | 31 |  | 32 | 34 | 25 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 |  |

**Code explanation**

* **Greedy.java**
  + **solve():** get initial board and goal board and create new board and new goal board by iterate 2 tiles by 2 tiles.
  + **ignoreTiles():** get a board from solve() and make all numbers negative except the two target numbers and return the board.
  + **addToOpen():**  add to open list.
  + **solution():** gets closed list and add to stack(solution list) and return them as a solution.
  + **getRestRows():** once complete to solve a first row, set the rest of the rows as a new start board and goal board.
  + **combine**(): combine with rows (total number of rows – 2) that solved with 2tile by 2tiles and the rest of rows.
* **State.java**
  + **State():** create an Object
    - State parent
    - int[][] board
    - Point blank
    - int gval //g score
    - int goalDistance
    - String direction //store tile number and direction for solution (e.g. 15 D)
  + **gethValues():** calculates linear Conflict
* **StateExpand.java**
  + **expandStates():** create new board that after a tile get slide depends where the blank is located
* **AStar.java**
  + **solve():** A\* algorithm; using Priority Queue for open list to keep list ascending order, based on heuristic score, in order to do that, I override compare() function to compare heuristic score. For closed list, I used HashSet to prevent storing same board, in order to do that, I override hashCode() and equals() to compare a board in Object.
* **Gvalue.java:** getting g score.
* **Manhattan.java**
  + **gethValue():** calculates manhattan distance.

**Ideas that didn’t work and did work**

I tried to solve a puzzle as a whole with A\* with h score (Manhattan Distance + Hamming Distance) + g score as heuristics first. It worked 3 by 3 puzzle and 4 by 4, but it takes more than 30 seconds to solve 4 by4. Switching data structure helped speed up my algorithm, for example, using Priority Queue instead of the regular queue and HashSet for closes list to prevent storing duplicates. Next, I tried using IDA\* to save some memory with h score (Manhattan Distance + Linear Conflict) + g score, it worked with 3 by 3 puzzle and 4 by 4 puzzle, but could not solve 5 by 5 puzzle. So, I switched approaching to 2 tiles by 2 tiles, I don’t know it is technically called “greedy search”, although the solution is not optimal, it worked up to 6 by 6 in less than 30 seconds, but from 7 by 7 it takes more than 1 minute or cannot even solve.

**Summary**

I don’t think there was an easy part, but the hard part was solving a large puzzle using only A\* or IDA\* algorithm or finding the optimal solution, it has to be efficient enough to solve over 5 by 5, so it definitely required more time trying to make it efficient and come up good heuristic. I ended up approaching 2 tiles by 2tiles (greedy search) with A\* and I didn’t have time to try another way, for example, try to solve row by row and column by column, or 2 tiles by 2 tiles with IDA\* algorithm, it could work up to 9 by 9.