CSCI 6511 Project3

G26063532   Zhihan Jiang

1. **Files**

   The folder submitted contains three files.

   solver.py, **linear.py**, and **n-puzzle.txt**

   "solver" is the solver of the game. "linear" is the heuristic component for solver while "n-puzzle" is the example input. You can replace the .txt file with your customized input but be sure to use the same name and put it at the same place.

2. **Execution**

   To run, use the command:

   $ python solver.py

   The program would go solve the game specified in the txt file. If it finds a path, the program would output the number of moves, time needed, and a sequence of operations to reach the goal from the given input. Each element in a sequence tell us if the empty cell should go which direction.

   For example, given the below input matrix, "UP" means

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

3. **Performance & Expectations**

   I am using linear conflicts plus Manhattan distance as my heuristic function. The framework is the classical A* so the program is not expected to run extremely fast especially fed with 15 puzzles. In fact, you can expect a result within one minute if the optimal solution requires no more than 35 moves.

   (The program takes all input matrix as legal as default)

   It's a pity but I have to say the program won't be able to solve a 5X5 input.

4. **Implementation & Lessons**

   I came across some crucial issues during my attempts to solve these puzzles. One of them is that how to maintain all those open nodes in the fringe. I was using list but that could hardly handle any game with more than around 12 moves. Then I turned to using heap sort in the final version which helped reduced runtime significantly.

   I did some research on n-puzzle and recognize that I better use IDA* for 15 and 24 version but I have no time to rewrite and refactor my codes. Lessons in classical A* consumed me a lot of time. Nevertheless, implementing the linear conflicts as originally defined in Mayer and Yung's *Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics** (1992) is quite satisfying. You can see more details in my linear.py which applies a recursion approach to compute these conflicts.