CSC3100 Assignment 3

Important Notes:

1. The assignment is an individual project, to be finished on one's own effort.
2. Submissions before the deadline 6pm Dec. 6, 2021 are treated as normal submissions. Submissions within one week after the deadline are treated as late submissions.
3. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism.

Marking Criterion:

1. The full score of the assignment is 100 marks.
2. Normal submission: 100 marks are given if a submission passes the test.
3. Resubmission: For normal submissions that fail the test, a resubmission is allowed within one week after the marking. 75 marks are given if the resubmission passes the test.
4. Late submission: 75 marks are given if it passes the test. No resubmission is allowed.
5. Zero mark is given if: there is no (normal or late) submission; both normal and late submissions fail the test; a normal submission fails the test and there is no resubmission; a late submission fails the test.
6. In case of identical copies, both submissions will be marked as zero. Similar rules apply to other cases of plagiarism found.

Running Environment:

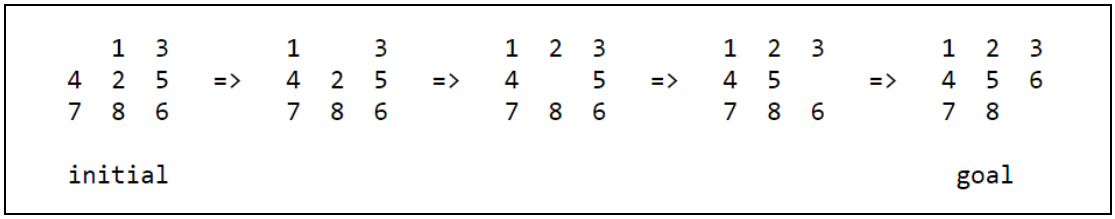
1. The submissions will be evaluated in Java environment under Linux platform.
2. The submission is acceptable if it runs in any of recent versions of Java SDK environment. These versions include from Java SE 8 to the most recent Java SE 17.
3. The submission is only allowed to import three packages of (java.lang.*; java.util.*; java.io.*) included in Java SDK. No other packages are allowed.
4. In the test, each program is required to finish within 5 seconds of time, with no more than 128MB memory.
5. Each student is free to test his/her program in the evaluation platform before the submission.

Submission Guidelines:

1. Detailed submission guideline will be given in a separate manual around Nov. 29.
2. Inconsistency with or violation from the guideline leads to marks deduction.

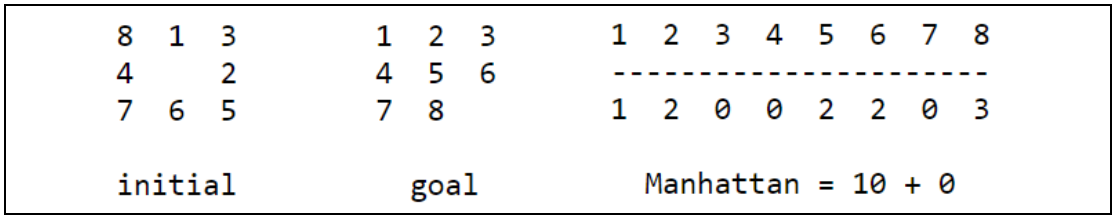
Problem Description:

Write a program to solve the 8-puzzle problem (and its natural generalizations) using the A* search algorithm. The 8-puzzle problem is a puzzle invented in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position to the goal position.



An algorithmic solution to the problem is based on the classical A* search algorithm (https://en.wikipedia.org/wiki/A*_search_algorithm). We define a state of the game to be the board position, the number of moves made to reach the board position, and the previous state. First, insert the initial state (the initial board, 0 moves, and a null previous state) into a priority queue. Then, delete from the priority queue the state with the minimum priority, and insert onto the priority queue all neighboring states (those that can be reached in one move). Repeat this procedure until the state dequeued is the goal state. The success of this approach hinges on the choice of priority function for a state. We consider the following priority function:

- **Manhattan priority function.** The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state.



There is a key observation here: to solve the puzzle from a given state on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using the Manhattan priority function. (This is true because each block must move its Manhattan distance from its goal position. Note that we do not count the blank tile when computing the Manhattan priorities.)

Consequently, as soon as we dequeue a state, we have not only discovered a sequence of moves from the initial board to the board associated with the state, but one that makes the fewest number of moves.

Functional Requirement

Write a program called **Puzzle8.java** that reads the initial board from an input file (file name specified by the user) and outputs a file (file name specified by the user) with a sequence of board positions that solves the puzzle in the fewest number of moves.

java Puzzle8 initial.txt moves.txt

The input file will consist of the 3-by-3 initial board position. The input format uses “0” to represent the blank square. As an example,

initial.txt
0 1 3 4 2 5 7 8 6

After running the program, the output file will be:

moves.txt
0 1 3 4 2 5 7 8 6 1 0 3 4 2 5 7 8 6 1 2 3 4 0 5 7 8 6 1 2 3 4 5 0 7 8 6 1 2 3 4 5 6 7 8 0

Program Template:

There is no program template for this assignment.

Appendix

1. initial.txt
2. moves.txt
3. astar_algorithm.pdf