CS420 -- Project 1: 8-Puzzle **Due: 2/9/2018 by 11:59pm**

Please use either Java, or C++ for implementation.

Project Description

The A* search can be used to solve the 8-puzzle problem. As described in the book, there are two candidate heuristic functions:

h1 = the number of misplaced tiles

h2 = the sum of the distances of the tiles from their goal positions

You are to implement the A* using both heuristics and compare their efficiency in terms of depth of the solution and search costs. The following figure (Figure 3.29 in the book) provides some data points that you can refer to. To test your program and analyze the efficiency, you should generate random problems (>100 cases) with different solution lengths. Please collect data on the different solution lengths that you have tested, with their corresponding search cost (# of nodes generate). A good testing program should test a range of possible cases (2 <= d <= 20). Note that the average solution cost for a randomly generated 8-puzzle instance is about 22 steps.

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ~~ITERATIVE DEEPENING SEARCH~~ and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

To help the instructor evaluate your program, the input of your program should be both: (1) a randomly generated 8-puzzle problem by your program; and (2) a specific 8-puzzle configuration entered through the standard input, which contains the configuration for only one puzzle, in the following format (where 0 represents the empty tile):

1 2 4 0 5 6 8 3 7

The goal state is:

0 1 2 3 4 5 6 7 8

* Please handle the input/output gracefully.

Note: the 8-puzzle states are divided into two disjoint sets, such that any state is reachable from any other state in the same set, which no state is reachable from any state in the other set. Before you solve a puzzle, you need to make sure that it's solvable. Here's how:

> **Definition:** For any other configuration besides the goal, whenever a tile with a greater number on it precedes a tile with a smaller number, the two tiles are said to be inverted.
>
> **Proposition**: For a given puzzle configuration, let N denote the sum of the total number of inversions. Then (N mod 2) is invariant under any legal move. In other words, after a legal move an odd N remains odd whereas an even N remains even. Therefore the goal state described above, with no inversions, has N = 0, and can only be reached from starting states with even N, not from starting states with odd N.

What to Submit?

Project report: (your approach + comparison of the two approaches + other analysis + findings), including a table similar to the Figure above, **without the effective branching factor**, **but with new columns about the average run time and the number of cases** you've tested with a specific length. (<= 3 pages, in pdf format).

Source code + README (how to compile or run your code).

Program output: three sample solutions with solution depths > 10. Your program should output each step from the initial state to the final state. For your testing purposes, you'll still need to generate 100 cases and document them.

Please create a folder called "lastname_firstname__420p1" that includes all the required files, from which, you should generate a zip file called "lastname_firstname_420p1.zip". For example, if Jane Doe was submitting a project, she would name the folder doe_jane_420p1. The resulting zip file would be named, doe_jane_420p1.zip. Submit this file via Blackboard before the due date.

No late submissions will be accepted.