CS4200 -- Project 1: 8-Puzzle
Use only Java, or C++ for implementation.

Project Description

The A* search can be used to solve the 8-puzzle problem. As described during lectures and 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 (modified 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 (>1000 cases, do not use the samples provided) with a variety of solution depths. Collect the data on the different solution depths that you have tested, with their corresponding search cost (# of nodes generated); and average the search costs by depth. Also, collect the including average run time per depth. A good testing program should test a range of possible cases (2 <= d <= 24).

Note that the average solution depth for a randomly generated 8-puzzle instance is approximately 22.

	A* Search Costs (nodes generated)	
d	h_1	h_2
2	6	6
4	13	12
6	20	18
8	39	25
10	93	39
12	227	73
14	539	113
16	1301	211
18	3056	363
20	7276	676
22	18094	1219
24	39135	1641

Comparison of the average search costs for the A* algorithm using heuristics h_1 and h_2. This data was averaged over 100 instances of the 8-puzzle for each depth.

User Interface: Provide at least two menu options for your program:

1) An option to generate a random solvable 8-puzzle problem (generated by your program) then solve it using both heuristics. It should output the optimal sequence of states that result from the search, the solution depth, the search cost for each heuristic, and the time it took to perform the search.
2) An option to input a specific 8-puzzle configuration. The input will contain the configuration for only one puzzle, in the following format **(where 0 represents the empty tile and the digits are separated by a space):**

   1 2 4 0 5 6 8 3 7

   This option should solve the entered puzzle and output the optimal sequence of states that result from the search, the solution depth, the search cost for each heuristic, and the time it took to perform each search.
   Your program must test the puzzle to be sure that it is solvable.
   You must handle the input/output gracefully—handle exceptions. If an unsolvable puzzle is enter, you must report it and prompt for another puzzle to be entered.

More about the problem: 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 is solvable. Here is 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?

1. **Project report:** (your approach + comparison of the two approaches + other analysis + findings), including a table similar to the Figure above, **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).
2. **Source code + README (how to compile or run your code).**
3. **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 >1000 cases and document them.

How to submit it

- Create a folder called "lastname_firstname__4200p1" that includes all the three required items listed above. You should generate a zip archive called "lastname_firstname_4200p1.zip" of that folder .
    - For example, if Jane Doe was submitting a project, she would name the folder doe_jane_4200p1. The resulting zip file would be named, doe_jane_4200p1.zip.
- Submit this file via Blackboard before the due date--**do not submit the project any other way.**

NO LATE SUBMISSIONS WILL BE ACCEPTED
DO NOT EMAIL YOUR SUBMISSION UNLESS ASKED TO
ONLY SUBMIT WORK THAT IS YOUR OWN.
PLAGIARISM WILL RESULT IN A FAILING GRADE FOR THE CLASS