CS4200 -- Project 1: 8-Puzzle
Use either 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 using both Tree-Search and Graph-Search algorithms You must 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 (>1000 cases, do not use the samples provided) with a variety of solution depths. Collect the data on the different solution lengths that you have tested, with their corresponding search cost (# of nodes generate); and average the search costs by depth. A good testing program should test a range of possible cases (2 <= d <= 24).
Note that the average solution cost for a randomly generated 8-puzzle instance is about 22 steps.

	Search Cost (Graph-search)	
d	A* (using h_1)	A* (using 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.

The goal state is: 0 1 2 3 4 5 6 7 8
You will need to write a simple UI to take user input. Your menu should offer the following:

1) Generate a randomly 8-puzzle problem
    a) Your agent will only work with an initial state that is solvable. Generate random states until one is found
    b) You must test the puzzle to be sure that it is solvable.
2) Enter a specific 8-puzzle configuration
    a) You should prompt the user to enter it. The will enter the configuration for only one puzzle, in the following format (where 0 represents the empty tile and the digits are separated by a space):
    b) 1 2 4 0 5 6 8 3 7

Either option should test with both heuristics
Your program must output for each heuristic, the sequence of states from the initial state to the goal, the search cost (nodes generated), the time spent on the search, and the path-cost of the solution.

You must handle the input/output gracefully (handle exceptions).

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?

1. **Project report:** (your approach + comparison of the two algorithms and heuristics + other analysis + findings), including a table similar to the Figure above for both A* on Graph-search and A* on tree-search, **but include new columns about the average run time and the number of cases** you've tested with a specific length. (at least 3 pages, in pdf format + a required cover page).
2. **Source code + README (how to compile or run your code). Be sure the readme states how to compile and run your code from the command line using g++ for C++, or javac for Java. Do not use namespace or package hierarchy.**
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 >400 cases and document them.
4. **Do not submit binary or jar files, only your ".cpp" or ".java" files**

How to submit it

- Create a folder called "lastname_firstname__4200p1" that includes all the required files, from which, you should generate a zip file called "lastname_firstname_4200p1.zip".
    - 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

READ THE SYLLABUS CONCERNING ACADEMIC INTEGRITY
NO EXCUSE WILL BE ACCEPTED