Project 1: 8-Puzzle

1. Description

The A* search can be used to solve the 8-puzzle problem. As described in the class, there are two candidate heuristic functions: (1) h1 = the number of misplaced tiles; (2) 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 provides some data points that you can refer to.

	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	3644035	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

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.

Input/output:

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

1 2 4
0 5 6
8 3 7

The goal state is:

0 1 2
3 4 5
6 7 8

Where 0 represents the empty tile.

Please handle the input/output gracefully.

Checking if 8-puzzle is solvable:

Note that 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.

Hint: use the fact that board positions are divided into two equivalence classes with respect to reachability: (i) those that lead to the goal position and (ii) those that lead to the goal position if we modify the initial board by swapping any pair of adjacent (non-blank) blocks. There are two ways to apply the hint:

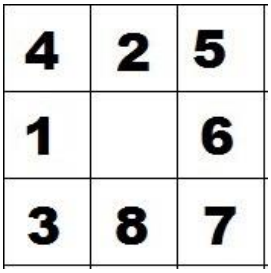
- Run the A* algorithm simultaneously on two puzzle instances - one with the initial board and one with the initial board modified by swapping a pair of adjacent (non-blank) blocks. Exactly one of the two will lead to the goal position.
- Derive a mathematical formula that tells you whether a board is solvable or not.

Linear Conflict + Manhattan Distance

Aside from 2 above common heuristics, linear conflict heuristic is very strong heuristics and said to be better than both. You are required to implement linear conflict heuristic and make comparisons. Below are explain about this heuristic:

Two tiles ‘a’ and ‘b’ are in a linear conflict if they are in the same row or column, also their goal positions are in the same row or column and the goal position of one of the tiles is blocked by the other tile in that row.

Let’s take the following example:



In this instance we see that tile 4 and tile 1 are in a linear conflict since we see that tile 4 is in the path of the goal position of tile 1 in the same column or vice versa, also tile 8 and tile 7 are in a linear conflict as 8 stands in the path of the goal position of tile 7 in the same row. Hence here we see there are 2 linear conflicts.

As we know that heuristic value is the value that gives a theoretical least value of the number of moves required to solve the problem we can see that one linear conflict causes two moves to be added to the final heuristic value(h) as one tile will have to move aside in order to make way for the tile that has the goal state behind the moved tile and then back resulting in 2 moves which retains the admissibility of the heuristic.

Linear conflict is always combined with the Manhattan distance to get the heuristic value of that state and each linear conflict will add 2 moves to the Manhattan distance as explained above, so the ‘h’ value for the above state will be

Manhattan distance + 2*number of linear conflicts

Manhattan distance for the state is: 10
Final h: 10 + 2*2= 14

Linear Conflict combined with Manhattan distance is significantly way faster than the heuristics explained above and 4 x 4 puzzles can be solved using it in a decent amount of time.

---

2. **What to Submit?**

   Note that this is individual exercises and should use Python.

- Project report: (your approach + comparison of the two approaches + other analysis + findings), including a table similar to the Figure above, without the

branching factor, but with new columns about the average run time and the number of cases you've tested with a specific length. Explain how to check if 8-puzzle is solvable

- Source code + README:
  - How to compile and run your code
  - Estimating the degree of completion level for each requirement.
- 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.

---

3. Grading policy?

No.	Description	Score
1	Successfully solve an input 8 puzzle	35
2	Testing 100 cases and make comparison	20
3	Checking if 8-puzzle is solvable	15
4	Linear conflict heuristic	10
5	Report and policy in “What to submit”	20
	Total	**100**