**Due: Week 7 before class (in two weeks)**

**Instruction:**

* **Submit the source codes together with this document to Moodle**
* Your source codes must contain your StudentId and Name
* You MUST follow the general program frameworks we studies in class

**Question 1 (See Exercise 1 of Chapter 5)**

Study the Python program 8-puzzle.py, which uses A\* search for solving the 8-puzzle problem that implements both the wrong tile heuristic (h1) and Manhattan distance heuristic (h2).

**Task 1: Complete the h2() function**

****

Note that the manhattan distance function is already completed for you, and you are welcome to call it in the code of h2,

**Task 2: Experiments**

Experiment with the following test cases and report the running time for both heuristics as well as whether the solution has been found for the following problems.

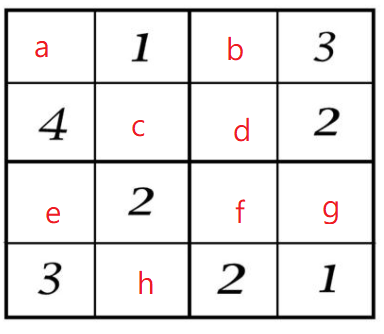
1. [1,0,3,4,2,5,6,7,8]
2. [5,1,3,4,0,8,7,6,2]
3. [6,4,3,1,0,8,5,2,7]
4. [4,7,2,8,0,1,3,6,5]
5. [1,7,4,2,3,8,0,6,5]

Submit your findings in the following format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | h1 (number of wrong tiles) | | h2 (Manhattan distance) | |
|  | Success? | Time (s) | Success? | Time (s) |
| 1. | Yes | 0.010693073272705078 | Yes | 0.00315802001953125 |
| 2. | Yes | 0.00729680061340332 | Yes | 0.0014297962188720703 |
| 3. | Yes | 0.7420070171356201 | Yes | 0.02639028549194336 |
| 4. | Yes | 19.011950969696045 | Yes | 0.2428819179534912 |
| 5. | Yes | 59.888928174972534 | Yes | 0.5798630714416504 |

**Question 2**

Complete the program (sudoku.py) to solve the 4\*4 Sudoku problem as shown below.



**Variable:** a,b,c,d,e,f,g,h

**Domain:** {1,2,3,4}

**Constraints:**

* Each smaller squares must contains 1 – 4:
* For example, constraint for the upper left squares can be wrriten as

sorted([a,1,4,c]) == [1,2,3,4]

(Alternatively, you can convert each four elements into a set, and check whether its length is 4)

* Each row must contain 1 – 4
* Each column must contain 1 - 4

**Bonus (Challenging) [10 bonuses]** (sudoku\_9.py)

(For those who want some real challenge.)

Rewrite the program to solve the following 9 \* 9 puzzle sudoku problem.

A picture containing text, crossword puzzle

Description automatically generated

Note: you may hard-code the puzzle into your code. E.g.

init\_state = [7, 0, 0, 0, 0, 0, 0, 1, 9,

4, 6, 0, 1, 9, 0, 0, 0, 0,

0, 0, 0, 6, 8, 2, 7, 0, 4,

0, 9, 0, 0, 0, 0, 0, 0, 7,

0, 0, 0, 3, 0, 0, 4, 0, 5,

0, 0, 6, 7, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0, 0,

2, 0, 0, 0, 7, 4, 0, 0, 0,

0, 0, 0, 2, 0, 0, 3, 0, 0]

(or equivalent)

**Notes:** the simple method we discussed above run too slowly for the 9 \* 9 puzzle. So in addition to what we have discussed, you may need to add extra constraints to speed up the search.

For example, if the first row is [7, 1, none, none, none, none, 1, 9], does it violate the constraint already? Do we really need to search its children even if there are still some "nones"?

**Hints**: add constraints to make sure each number in (1 – 9) only occur once in each row / col / square.

Ideally, your code should find the answer in less than 1 minute.